



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

# Efficient Private Function Evaluation

at the Department of Computer Science  
of the Technical University of Darmstadt

## Doctoral Thesis

submitted in fulfillment of the requirements for the degree of  
Doctor of Engineering (Dr.-Ing.)

by

Ágnes Kiss, M.Sc.

born in Szolnok, Hungary

Advisors: Prof. Dr.-Ing. Thomas Schneider  
Prof. Dr. N. Asokan

Date of submission: 01.08.2020

Date of defense: 25.09.2020

D 17  
Darmstadt, 2020

This document was published by tuprints, an e-publishing service of the Technical University of Darmstadt.

<http://tuprints.ulb.tu-darmstadt.de>  
[tuprints@ulb.tu-darmstadt.de](mailto:tuprints@ulb.tu-darmstadt.de)

Please cite this document as:

URN: urn:nbn:de:tuda-tuprints-17496

URL: <http://tuprints.ulb.tu-darmstadt.de/id/eprint/17496>

The publication is under the Copyright law of Germany.

---

## **Erklärung**

Hiermit versichere ich, Ágnes Kiss, M.Sc., die vorliegende Doctoral Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Darmstadt, 31.07.2020

---

Ágnes Kiss, M.Sc.

---

## Abstract

Private function evaluation (PFE) allows two or more parties to jointly compute a private function of one of the parties on the private inputs of the other parties securely. PFE can provide a viable solution in a scenario where a server holding a function that is his intellectual property or trade secret would like to compute this function on a client's sensitive input, in a privacy-preserving manner. In recent years, multiple approaches have been presented for PFE. These approaches are based on techniques that are used for secure function evaluation (SFE) as well, where parties jointly compute the result of a publicly known function on their private inputs while keeping their inputs private. These techniques include homomorphic encryption, garbling techniques and secret sharing.

In this thesis, we present results that improve the practicality of private function evaluation where nothing about the function is revealed except its (maximum) size. The contributions are split in two parts based on the underlying function representation as follows:

**PFE of Boolean circuits.** Boolean circuits can represent any Boolean function and are therefore generic. PFE can be realized with the secure evaluation of a so-called universal circuit (UC) that can be programmed with a set of program bits to compute any function up to a given size  $n$ . We are the first to implement asymptotically optimal UCs with size  $\Omega(n \log n)$  that have been proposed by Valiant (STOC'76). We improve their concrete size by providing optimizations and show that PFE with UCs is efficient for realistic circuit sizes with hundreds of thousands of gates. We identify that the bottleneck of our PFE implementations is memory consumption, and therefore, design an algorithm that utilizes only  $\mathcal{O}(n)$  memory at each step of the protocol execution. PFE of Boolean circuits can also be realized using additively homomorphic encryption and standard SFE techniques. This linear-complexity approach was first introduced by Katz and Malka (ASIACRYPT'11). We have shown that their protocol is practical and that our optimized implementation achieves PFE with the best performance to date for circuits starting already at a few thousand gates.

This part of the thesis is based on the following four publications:

- [KS16] Á. KISS, T. SCHNEIDER. “**Valiant’s Universal Circuit is Practical**”. In: 35. *Advances in Cryptology – EUROCRYPT’16*. Vol. 9665. LNCS. Full version: <https://ia.cr/2016/093>. Code: <https://encrypto.de/code/UC>. Springer, 2016, pp. 699–728. CORE Rank A\*. Appendix A.
- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: 23. *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Full version: <https://ia.cr/2017/798>. Code: <https://encrypto.de/code/UC>. Springer, 2017, pp. 443–470. CORE Rank A. Appendix B.
- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Springer. Online version: <https://ia.cr/2019/348>. Code: <https://encrypto.de/code/UC>, pp. 1216–1271. CORE Rank A\*. Appendix C.



- 
- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: 25. *European Symposium on Research in Computer Security (ESORICS’20)*. Vol. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://encrypto.de/code/linearPFE>. Springer, 2020, pp. 401–420. CORE Rank A. Appendix D.

**PFE of decision trees.** Decision trees are a machine learning model that allow for efficient classification based on a set of input features. Privacy-preserving decision tree evaluation has been considered in the literature, both using homomorphic encryption and garbling techniques. We systematically analyze existing protocols, identify the sub-protocols of private decision tree evaluation, and present novel combinations that result in better communication and computation tradeoff than previous methods.

This part of the thesis is based on the following systematization of knowledge (SoK) publication:

- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2019.2* (2019). De Gruyter Open. Online version: <https://ia.cr/2018/1099>. Code: <https://encrypto.de/code/PDTE>, pp. 187–208. CORE Rank B. Appendix E.

This thesis presents results that contribute to making private function evaluation (PFE) efficient and practical for deployment in use cases such as privacy-preserving classification for medical diagnostics and privacy-preserving insurance rate calculation.

---

## Zusammenfassung

Private Funktionsevaluierung (PFE) ermöglicht es zwei oder mehr Parteien gemeinsam eine private Funktion, die eine der Parteien kennt, auf den privaten Eingaben der anderen Parteien zu berechnen. Mit PFE lassen sich Szenarien ermöglichen, in denen eine private Funktion eines Servers, die ein geistiges Eigentum oder ein Geschäftsgeheimnis beinhalten kann, auf den sensitiven Daten eines Clients berechnet wird. In den letzten Jahren wurden mehrere PFE Ansätze vorgestellt. Diese basieren auf bekannte Techniken aus der sicheren Funktionsevaluierung (SFE), bei der die Parteien gemeinsam eine öffentlich bekannte Funktion auf ihren privaten Eingaben berechnen, ohne dass eine der Parteien die Eingaben einer anderen Partei lernt. Zu diesen Techniken zählen homomorphe Verschlüsselung, Garbling-Techniken und Secret Sharing.

Diese Dissertation zeigt praktisch anwendbare Lösungen für private Funktionsevaluierung, bei denen ausschließlich die (maximale) Größe der Funktion offenbart wird. Die Beiträge sind in zwei Abschnitte aufgeteilt, die sich aus der Art ergeben, wie die Funktion repräsentiert wird:

**PFE von Booleschen Schaltkreisen.** Boolesche Schaltkreise sind eine generische Option, Funktionen zu repräsentieren, da sie jede Boolesche Funktion darstellen können. PFE kann durch sichere Funktionsevaluierung eines universellen Schaltkreises (UC) ermöglicht werden, der mittels Programmierbits zur Berechnung einer beliebigen Funktion von maximaler Größe  $n$  programmiert werden kann. Wir sind die Ersten, die einen asymptotisch optimalen UC der Größe  $\Omega(n \log n)$  implementiert haben, der auf Valiant's Konstruktion (STOC'76) basiert. Wir optimieren die konkrete Größe von UCs und zeigen damit, dass PFE mittels UCs effizient für realistische Schaltkreis-Größen mit mehreren Millionen Gates umsetzbar ist. Wir identifizieren, dass der Flaschenhals unserer PFE Implementierungen der Speicherverbrauch ist, weswegen wir einen Algorithmus entworfen haben, der nur  $\mathcal{O}(n)$  Speicher in jedem Protokollschritt benötigt. PFE von Booleschen Schaltkreisen kann alternativ auch mit additiver homomorpher Verschlüsselung und Standard SFE Techniken realisiert werden. Katz und Malka (ASIACRYPT'11) sind die Ersten, die diesen Ansatz mit linearer Komplexität vorgestellt haben. Wir haben gezeigt, dass dieser Ansatz praktikabel ist und dass unsere optimierte Implementierung die bis zum heutigen Tag performanteste PFE Lösung für Schaltkreise bereits ab einer Größe von einigen tausend Gattern ist.

Dieser Abschnitt der Dissertation basiert auf folgenden vier Veröffentlichungen:

- [KS16] Á. KISS, T. SCHNEIDER. “**Valiant's Universal Circuit is Practical**”. In: 35. *Advances in Cryptology – EUROCRYPT'16*. Bd. 9665. LNCS. Full version: <https://ia.cr/2016/093>. Code: <https://encrypto.de/code/UC>. Springer, 2016, S. 699–728. CORE Rank A\*. Appendix A.
- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: 23. *Advances in Cryptology – ASIACRYPT'17*. Bd. 10625. LNCS. Full version: <https://ia.cr/2017/798>. Code: <https://encrypto.de/code/UC>. Springer, 2017, S. 443–470. CORE Rank A. Appendix B.

- 
- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Springer. Online version: <https://ia.cr/2019/348>. Code: <https://crypto.de/code/UC>, S. 1216–1271. CORE Rank A\*. Appendix C.
- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: 25. *European Symposium on Research in Computer Security (ESORICS’20)*. Bd. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://crypto.de/code/linearPFE>. Springer, 2020, S. 401–420. CORE Rank A. Appendix D.

**PFE von Entscheidungsbäumen.** Entscheidungsbäume sind ein Modell des maschinellen Lernens, die eine effiziente Klassifizierung anhand von einer Menge von Eingabe-Eigenschaften ermöglichen. Die privatsphäre-freundliche Evaluierung von Entscheidungsbäumen wurde in der Literatur mit homomorpher Verschlüsselung und Garbling-Techniken betrachtet. Wir analysieren systematisch existierende Protokolle, identifizieren die Unterprotokolle zur privaten Auswertung von Entscheidungsbäumen, und präsentieren neuartige Kombinationen, die einen besseren Abgleich von Kommunikation und Laufzeit aufweisen als bisher bekannte Methoden.

Dieser Abschnitt der Dissertation basiert auf folgender Systematization of Knowledge (SoK) Publikation:

- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2019.2 (2019). De Gruyter Open. Online version: <https://ia.cr/2018/1099>. Code: <https://crypto.de/code/PDTE>, S. 187–208. CORE Rank B. Appendix E.

Diese Dissertation präsentiert Ergebnisse, die private Funktionsevaluierung (PFE) effizient umsetzbar und praktikabel machen und für praktische Anwendungsfälle wie privatsphäre-freundlicher medizinischer Diagnostik und privaten Berechnungen von Versicherungstarifen eingesetzt werden können.

---

## My Contributions

During my work on this thesis, I had the chance to collaborate with excellent researchers worldwide (Thomas Schneider, N. Asokan, Jian Liu, Masoud Naderpour and Deevashwer Rathee) as well as with outstanding students at the Technical University of Darmstadt (Daniel Günther, Masaud Y. Alhassan and Marco Holz). I thank all my co-authors for their contributions and for our fruitful discussions on the papers this thesis is based on.

Chapter 2 is based on [KS16], [GKS17], [AGKS20], and [HKRS20]. [KS16] was joint work with Thomas Schneider, where I significantly contributed to all aspects of the publication, i.e., the clear description, design and implementation of the universal circuit construction. [GKS17] is a paper based on the outstanding B.Sc. thesis of Daniel Günther, who was awarded with 1st prize in the category of B.Sc. theses at the CAST Award in IT Security (CAST Förderpreis IT Sicherheit). My contribution was the initial idea, the thesis co-supervision and a new construction present in our paper, whereas Daniel provided the more formal description of the construction and implemented the improved universal circuit. The journal article [AGKS20] summarizes and extends the results of [KS16; GKS17]. The design of our scalable universal circuit construction is a result of my fruitful discussions with Thomas Schneider and Masaud Y. Alhassan, whose M.Sc. thesis, co-supervised by me, has been a base for Section 5.4 of the paper. He has provided the implementation of this scalable construction, while Daniel and I have performed the implementation and evaluation of the recent improvements in the field. Moreover, I revised the texts and performed the evaluation and comparison of all methods. [HKRS20] is based on the M.Sc. thesis of Marco Holz (graded *very good*), co-supervised by me and Thomas Schneider. All co-authors contributed to the discussions that resulted in this paper. Marco Holz provided the main contribution, i.e., several optimizations along with the most efficient implementation of linear-complexity private function evaluation to date. Deevashwer Rathee helped with utilizing state-of-the-art homomorphic encryption schemes.

Chapter 3 is based on [KNL<sup>+</sup>19], a joint work with Masoud Naderpour, Jian Liu, N. Asokan and Thomas Schneider. All of us contributed to the fruitful discussions that resulted in this paper. I contributed to the description of all methods and provided the implementation for the sub-protocols based on lifted ElGamal homomorphic encryption, while Masoud has implemented the sub-protocols that use garbling techniques and Paillier or DGK homomorphic encryption. I performed the evaluation and provided the discussions of the results.

---

## Acknowledgments

This thesis is the result of five years of research in the ENCRYPTO group at TU Darmstadt. During this long and challenging journey in a town completely new to me, I have been supported by many people whom I would like to thank here.

Firstly and most importantly, I would like to thank my advisor, Thomas Schneider, who has been a great mentor to me throughout these years. I am incredibly grateful for the chance to be part of the ENCRYPTO group - an opportunity that allowed me to grow professionally in many directions: I have learned the essence of meaningful research and good scientific practice, had the chance to review papers, supervise undergraduate theses, and learn about project management. I have also grown emotionally by pushing my limits, learning to ask meaningful questions and not giving up when debugging universal circuits seemed to be an endless fight. Thomas has always answered to my (many many) questions and requests promptly - so much so that I sometimes wrote an email in the evening but sent it to him only the next day during work hours. From him, I have received not only excellent professional guidance but also personal support whenever it was necessary, for which I am very grateful.

It was a great honor to have the chance to collaborate with N. Asokan on multiple occasions and to have him as my external advisor during the final process. I would also like to thank Marc Fischlin, Jan Peters and Felix Wolf for agreeing to be on my Ph.D. defense committee and initiating interesting discussions during the disputation.

Being part of the ENCRYPTO group for the last few years allowed me to see it grow with more and more talented fellow students joining. Understandably, we have supported each other throughout the years both professionally and emotionally. I am thankful for the support of my colleagues with whom I have enjoyed exploring new research directions as well as tourist attractions (at least when it was still possible): Michael Zohner, Daniel Demmler, Christian Weinert, Oleksandr Tkachenko, Amos Treiber, Hossein Yalame, Daniel Günther, Helen Möllering, Marco Holz and Raine Nieminen. I am also very grateful for the support I have received from our team assistants: Karina Köhres, Heike Meissner, Melanie Schöyen and Petra Fuhrmann. I enjoyed spending time with everyone in the group during our lunch and coffee breaks and also outside of work.

I was fortunate enough to collaborate with excellent researchers throughout my Ph.D. journey, during which I have seen their examples of good scientific practice and communication. I would like to thank all of them for their time and devotion to our projects: Juliane Krämer, Pablo Rauzy, Jean-Pierre Seifert, Jian Liu, N. Asokan, Benny Pinkas, Daniel Günther, Oliver Schick, Kimmo Järvinen, Olexandr Tkachenko, Zheng Yang, Masoud Naderpour, Lukas Scheidel, Susanne Felsen, Christian Weinert, Masaud Y. Alhassan, Marco Holz, Deevashwer Rathee, Johannes Buchmann, Ghada Dessouky, Tommaso Frassetto, Ahmad-Reza Sadeghi, Giulia Traverso and Shaza Zeitouni.

I have received immense support from my friends throughout these years. In Darmstadt, I was lucky enough to make many new friends and would like to thank them for making me

---

feel home there. Specifically, I would like to thank Karina Köhres for all our long discussions and dinners, Nikolay Matyunin for making many theatre or cinema visits, dance classes and dinners fun and memorable, Wen Wang for her strength, optimism and our experience with the duck. I thank Jibesh Patra, Supriti Sinhamahapatra and Ahana Patra for their positive energy and parenting example, for all the trips we have made together and all the photos that were shot by our hobby photographer. I would also like to thank Peter Merz, Marija Selakovic, Jovan Krunić, Nathalie Brunner, Andrew Habib and Lydia Gad for the fun weekend visits and delicious foods we have enjoyed together. I thank Hanne Weismann and Matt Geddes for the crazy boardgaming months where I have learnt many and more amazing games and for our spontaneous trip to the big bench. I would like to thank Giorgia Marson for the amazing months spent together in the crypto-apartment and for the chance to try her potato pizza. I am also grateful for having had the opportunity to share many lunch breaks with Juliane Krämer, to learn Alchemists from Carlos García, to experience Tommaso Gagliardoni's great vibe, and to share many joyful moments and meaningful discussions with Giulia Traverso, Ghada Dessouky and Shaza Zeitouni during our CROSSING collaboration. I also thank Clara Paglialonga for taking a major role in „saving my life” at that waterfall in Hong Kong and for becoming my friend in Darmstadt, Jacqueline Brendel for being a great friend and the first person to babysit Anna, and Sogol Mazaheri for all our meaningful discussions. Besides my new friends in Darmstadt, I have received immense support from my old friends (both from Hungary and from the EIT Digital Master School). I would like to thank Réka Vidra, Judit Sztítás, Ági Dobos, Kati Hajdú-Szücs, Dalma Héricsz, Máté Horváth, Adri Gerencsér, Rita Varga, Nurul Momen and Rahnuma Tasneem Senjuti for always being there for me. I also thank Rainer and Ági Matthes for our amazing long weekends near Altdöbern and for always believing that my German will eventually be good enough to talk about anything. We have spent many hours practicing on the phone since I left Berlin.

Last but not least, I would like to thank my family for supporting me in my journey. My parents and my sister, Judit, have always been my first line of defense when I was in need of emotional support. Luckily, my family became bigger in the past few years, so besides my parents and Judit, I would like to thank Martin, my grandmothers, aunts and uncles, cousins as well as my Romanian family - Mami, Tati, Deni, Buni, Bunica and Unchiu for always being supportive during these times even when it has been difficult to be far away from each other. I am grateful for the visits, the amazing food and the great trips we had together, but most importantly, I am forever thankful for their unconditional love.

The most support, of course, I have received from my husband, Cris, who has been sharing this journey with me. He has been the person to sober me up in times when I felt lost and the person supporting me in every little step of my journey, not only emotionally but also with home catering. I thank you for being there for me throughout these years and will never forget the one time our place „exploded” with both of us having a big deadline at the same time. Last but not least, I would like to thank our little daughter Anna: I thank you for making me smile whenever I needed to cheer up, for becoming an amazing sleeper after only a few months and for allowing me to write my thesis from home-office (mostly) without having to worry about what you and Apa are up to in the other room.

# Contents

---

<b>Abstract</b>	<b>III</b>
<b>Zusammenfassung</b>	<b>V</b>
<b>My Contributions</b>	<b>VII</b>
<b>Acknowledgments</b>	<b>VIII</b>
<b>Contents</b>	<b>X</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Use Cases . . . . .	3
1.2 Thesis Outline . . . . .	4
1.3 Open Access . . . . .	5
<b>2 Private Function Evaluation of Boolean Circuits</b>	<b>6</b>
2.1 Our Contributions . . . . .	6
2.2 Related Work . . . . .	9
<b>3 Private Function Evaluation of Decision Trees</b>	<b>15</b>
3.1 Our Contributions . . . . .	16
3.2 Related Work . . . . .	17
<b>4 Conclusion and Future Work</b>	<b>23</b>
4.1 Summary . . . . .	23
4.2 Future Work . . . . .	24
<b>Bibliography</b>	<b>26</b>
<b>Lists</b>	<b>34</b>
<b>Curriculum Vitae</b>	<b>37</b>
<b>Appendices</b>	<b>41</b>
<b>A Valiant’s Universal Circuit is Practical (EUROCRYPT’16)</b>	<b>41</b>

<b>B</b>	<b>More Efficient Universal Circuit Constructions (ASIACRYPT'17)</b>	<b>72</b>
<b>C</b>	<b>Efficient and Scalable Universal Circuits (JoC'20)</b>	<b>101</b>
<b>D</b>	<b>Linear-Complexity Private Function Evaluation is Practical (ESORICS'20)</b>	<b>158</b>
<b>E</b>	<b>SoK: Modular and Efficient Private Decision Tree Evaluation (PoPETs'19)</b>	<b>179</b>



# 1 Introduction

---

Privacy is the ability of an individual to seclude themselves or their personal information, and thereby express themselves selectively. Data privacy is a branch of data security dealing with the proper handling of data, i.e., with how data is collected and stored, whether data can be shared with third parties, and how to comply with regulatory restrictions. Over two third of all countries include statements regarding privacy protection in their constitution, and over half of all countries have some form of data protection laws<sup>1</sup>. These help to ensure that personal data is protected correctly and thereby help to protect the general right to privacy of people.

Data privacy has recently gained increasing attention due to recent data privacy laws such as the European Union’s law on data protection and privacy, i.e., the General Data Protection Regulation (GDPR) [Eur16] that is in place since May 2018, or the California Consumer Privacy Act (CCPA) that became effective in January 2020. Due to these regulations, entities can no longer share or distribute private data of their customers, and therefore, computations on data from multiple sources require privacy-preserving technologies. These technologies are emerging and gaining attention as companies are bound to protect their customers’ data.

In many cases, applications such as data analytics and privacy-preserving machine learning require data to be gathered from multiple data providers who are now bound to protect the privacy of their clients and cannot share their data in clear. Therefore, companies worldwide such as Cybernetica<sup>2</sup> (Estonia), Partisia<sup>3</sup> (Denmark), Inpher<sup>4</sup> (Switzerland), Galois<sup>5</sup> (USA) and Unbound<sup>6</sup> (Israel) provide privacy-preserving solutions for use cases where the data to be computed on is supposed to be kept private.

The two main approaches for privacy-preserving computation include *homomorphic encryption* (HE) where computations directly performed on encrypted data are reflected on the plaintext after decryption and *secure multi-party computation* (SMPC), also referred to as *secure function evaluation* (SFE), that allows multiple mutually distrusting parties to jointly compute a function on their private inputs securely. According to the underlying security model, one or more of the computing parties are considered to be either passive/semi-honest (following the protocol but trying to learn secrets from observations) or malicious (deviating

---

<sup>1</sup><https://privacyinternational.org/explainer/56/what-privacy>

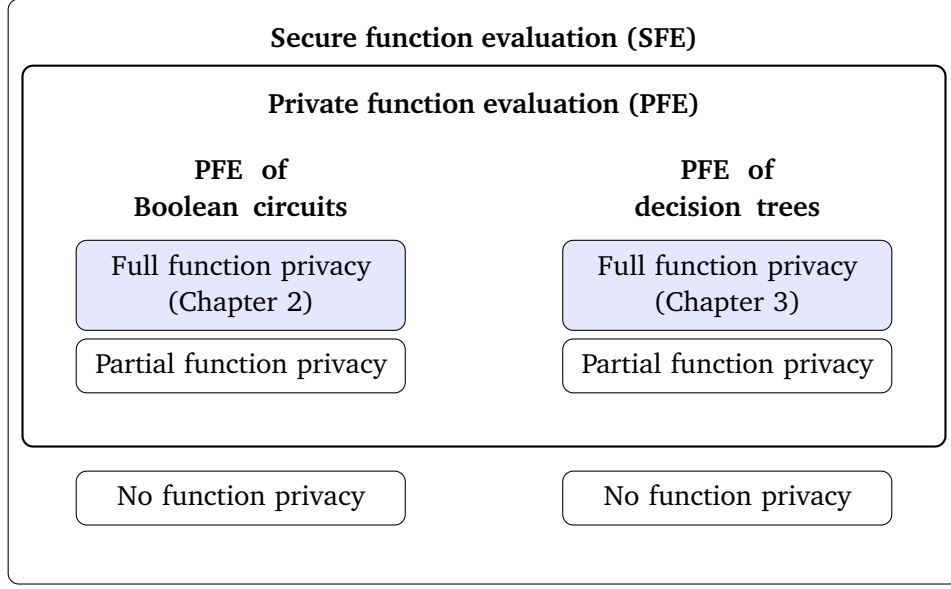
<sup>2</sup><https://cyber.ee/products/secure-analytics/>

<sup>3</sup><https://partisia.com/secure-statistics/>

<sup>4</sup><https://www.inpher.io/>

<sup>5</sup><https://galois.com/>

<sup>6</sup><https://www.unboundtech.com/>



**Figure 1.1:** Variants of secure function evaluation (SFE) and private function evaluation (PFE).

from the protocol execution). Tremendous improvements have been made in these fields recently, resulting in reasonable tradeoffs between privacy and efficiency.

However, most deployed solutions do not tackle the challenge when the service provider’s function itself – which can be their intellectual property or trade secret – is supposed to be kept private from the client, while the client’s data is still to be handled in a privacy-preserving manner. Private function evaluation (PFE) allows for mutually distrusting parties to compute a private function provided by one of the parties on the private inputs of the other parties. This can be achieved either by using fully homomorphic encryption (FHE) or by applying partially homomorphic encryption (HE) or techniques based on SFE. As FHE (HE that supports both additive and multiplicative operations) of complex functions is currently not a competitive approach due to its computational complexity, solutions based on HE and techniques used in SFE protocols are more promising and have gained increasing interest in recent years.

Private function evaluation can be categorized based on multiple aspects depicted in Figure 1.1. For instance, the function can be represented as a Boolean circuit [AF90; KS08a; MS13; MSS14; KS16; LMS16; GKS17; BBKL19; ZYZL19; AGKS20; BBKL20; LYZ<sup>+</sup>20], or it can be a decision tree for privacy-preserving classification [BPSW07; BPTG15; WFNL16; TMZC17; JS18; LZS18; ALR<sup>+</sup>19; CDH<sup>+</sup>19; KNL<sup>+</sup>19; LSC<sup>+</sup>19; TKK19; ZDW19; LCL<sup>+</sup>20; LSZ<sup>+</sup>20; TBK20] or its generalization, a branching program, also called a binary decision diagram (BDD) [BPSW07; IP07; BFK<sup>+</sup>09; BFL<sup>+</sup>11; Sch11]. Private neural network inference has also been considered [SS08], but most privacy-preserving solutions reveal the structure of the neural network as it is not considered to reveal much about the model itself [LJLA17; MZ17; BDK<sup>+</sup>18; JVC18; MR18; RWT<sup>+</sup>18; RRK18; EPI19; RSC<sup>+</sup>19; MLS<sup>+</sup>20].

Treiber et al. [TMW<sup>+</sup>20] propose private evaluation of sum-product networks (SPNs) where the structure is hidden using so-called random tensorized SPNs [PVS<sup>+</sup>19] to avoid information leakage about the training data. Naturally, PFE of Boolean circuits can be utilized in any use case but is computationally heavier for certain use cases than approaches designed specifically for decision trees. Moreover, different levels of the privacy of the function may be required for different applications. Generally, PFE with *full function privacy* hides the function  $f$  within all possible functions up to a given size, while so-called semi-private function evaluation (semi-PFE) with *partial function privacy* hides the function within a predefined group of functions (e.g., within the group of functions with the same [PKV<sup>+</sup>14] or similar [PSS09; KKW17] topology, or within a predefined set of functions [Kol18]). In this thesis, we investigate private function evaluation with full function privacy, both of Boolean circuits and of decision trees.

### 1.1 Use Cases

Private function evaluation (PFE) has several applications where the private functionality is held by one of the participants (usually a service provider) that is to be computed on the private input of the other participant(s) (usually a client or multiple clients). We review a few applications of PFE below.

**Financial applications.** In financial applications of PFE, the function is the intellectual property of one of the participants (i.e., a bank or an insurance company), and therefore, should be kept private from the other parties (i.e., the clients). Current solutions simply send the client's data to the function holder to perform computations on, and thereby do not handle their data privately. PFE protects the client's input as well as the function holder's function while allowing the parties to compute the result collaboratively. *Privacy-preserving checking for credit worthiness* [FAZ05] was one of the first proposed applications of PFE. Here, both the loaner's function and the loanee's data are kept private during the computation. We have proposed and implemented a prototype for *privacy-preserving insurance rate calculation* in [GKSS19] for car insurance companies who want to calculate user-specific tariffs based on potentially private inputs of their clients while not revealing their policies for these calculations.

**Proprietary software.** In applications where the service provider's program is its sensitive business information, PFE can provide a privacy-preserving solution. *Privacy-preserving intrusion detection* has been proposed in [NSMS14] where both the server's signatures and intrusion detection algorithm and the sensitive data of the client are to be protected. *Attribute-based access control* can also be enhanced using PFE to protect access control policies alongside sensitive credentials [FAL06; FLA06].

**Privacy-preserving classification.** Machine learning models are often required to be kept private as they may be the intellectual property of the model provider and moreover, it may leak information about the sensitive training data. *Privacy-preserving evaluation of diagnostic programs* such as medical diagnostics [BFK<sup>+</sup>09; TASF09; BFL<sup>+</sup>11] have been considered where both the client’s sensitive medical data and the diagnostic method are protected. Other classification use cases such as *malware and text classification* [AM18; RMD18] can also be performed in a privacy-preserving manner, protecting the classification algorithm from the client as well as the client’s sensitive data from the service provider.

**Databases.** *Private database management systems (DBMSs)* such as Blind Seer [PKV<sup>+</sup>14; FVK<sup>+</sup>15] make use of PFE to evaluate private queries in the form of an SQL statement on private data. Blind Seer uses semi-private function evaluation that does not hide the topology of the circuit, but their protocol could be extended by using PFE of Boolean circuits with full function privacy to provide full privacy of the queries.

## 1.2 Thesis Outline

Different use cases of private function evaluation (PFE) benefit from different function representations and different levels of privacy for the function. In this thesis, we consider methods that provide *full function privacy* and describe our contributions based on two main function representation methods as shown in Figure 1.1:

**Chapter 2:** Any Boolean function can be represented as a *Boolean circuit*, and therefore, this function representation is generic and can be utilized in any application scenario. Our work has shown that PFE with the secure evaluation of so-called universal circuits (UCs), that was believed to be highly inefficient, can become practical for realistic circuit sizes. We have brought Valiant’s construction [Val76] proposed four decades ago into practice and showed the potential of this method. Our work [KS16; GKS17; AGKS20] has inspired a new line of research on PFE [ZYZL19; LYZ<sup>+</sup>20] based on UCs that led to significant improvements of its size. After having shown the practicality of PFE using SFE of UCs, we have shown in [HKRS20] that linear-complexity PFE protocols using HE [KM11; MS13; BBKL19] can outperform this approach and achieve PFE with even more practical performance. We describe our contributions to this research field in Section 2.1 and position our work in the context of general-purpose private function evaluation in Section 2.2.

**Chapter 3:** Some applications such as diagnostic programs and classification methods naturally benefit from representing the function as a *decision tree*. These functions can be evaluated privately without having to be transformed into a less efficient Boolean circuit representation. In [KNL<sup>+</sup>19], we have analyzed protocols based on homomorphic encryption (HE) and secure function evaluation (SFE) for private decision tree evaluation, identified their sub-protocols and provided novel combinations of them that allow for better tradeoffs between communication and computation. We describe

our contributions in private decision tree evaluation in Section 3.1, and position our work in the context of related works in Section 3.2.

Finally, in **Chapter 4**, we conclude this thesis and give directions for future work on efficient private function evaluation.

### 1.3 Open Access

All papers in this thesis are available as open access via the Cryptology ePrint Archive<sup>7</sup> and come with open-source implementations. This allows other researchers to fairly compare with and extend our results and use our implementations to further improve the practicality of private function evaluation (PFE).

---

<sup>7</sup><https://eprint.iacr.org/>

## 2 Private Function Evaluation of Boolean Circuits

---

Private function evaluation (PFE) can be realized based on the Boolean circuit representation of the function. As any Boolean function can be represented as a Boolean circuit, this approach is *generic* and can be applied in any use case of PFE. The challenge of PFE of Boolean circuits with full function privacy is to hide the gates' functionality as well as the topology of the Boolean circuit. The latter imposes a large overhead compared to secure function evaluation (SFE) or secure multi-party computation (SMPC) protocols where the function to be computed is publicly known to all participants, and only the inputs are kept private.

Approaches for Boolean circuit-based PFE can be grouped into two categories: Linear-complexity homomorphic encryption (HE)-based approaches [KM11; MS13; MSS14; BBKL20; HKRS20] and approaches based on mostly symmetric cryptography with superlinear complexity [KS08a; MS13; KS16; GKS17; BBKL19; ZYZL19; AGKS20; LYZ<sup>+</sup>20] in the size of the private circuit. In this chapter, we describe our contributions to both approaches of Boolean circuit-based PFE in Section 2.1 and position our results in the literature in Section 2.2.

### 2.1 Our Contributions

In this section, we describe our contributions to PFE of Boolean circuits, both using universal circuits (UCs) (Section 2.1.1) and using homomorphic encryption (HE) (Section 2.1.2).

#### 2.1.1 PFE using Secure Function Evaluation (SFE) of Universal Circuits (UCs)

Private function evaluation (PFE) based on the Boolean circuit representation of the private function  $f$  can be realised by securely evaluating a so-called universal circuit. A universal circuit (UC) is a Boolean circuit  $UC$  that can be programmed to compute any Boolean circuit  $C$  up to a given size  $n$ , i.e., there exist programming bits  $p$  such that  $UC(p, x) = C(x)$  for any input  $x$ . Securely evaluating such a UC as a public function using SFE protects the privacy of the inputs  $p$  and  $x$  and therefore, directly allows for private function evaluation.

In 1976, Valiant [Val76] proposed a UC construction with two variants of sizes  $\sim 5n \log_2 n$  and  $\sim 4.75n \log_2 n$  in the size of the simulated circuit  $n$ , and has shown that the asymptotic lower bound on the size of a universal circuit is  $\Omega(n \log n)$ . Universal circuits (UCs) have been a mere theoretical concept until 2008, when Kolesnikov and Schneider [KS08a] proposed

and implemented their UC construction of asymptotic size  $\mathcal{O}(n \log^2 n)$ , and UC-based PFE with the same asymptotic complexity.

This thesis has significantly contributed to UC-based PFE with the following three publications that can be found in Appendices A, B, and C:

- [KS16] Á. KISS, T. SCHNEIDER. “**Valiant’s Universal Circuit is Practical**”. In: 35. *Advances in Cryptology – EUROCRYPT’16*. Vol. 9665. LNCS. Full version: <https://ia.cr/2016/093>. Code: <https://encrypto.de/code/UC>. Springer, 2016, pp. 699–728. CORE Rank A\*. Appendix A.
- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: 23. *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Full version: <https://ia.cr/2017/798>. Code: <https://encrypto.de/code/UC>. Springer, 2017, pp. 443–470. CORE Rank A. Appendix B.
- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Springer. Online version: <https://ia.cr/2019/348>. Code: <https://encrypto.de/code/UC>, pp. 1216–1271. CORE Rank A\*. Appendix C.

Valiant’s asymptotically optimal universal circuit (UC) with size  $\sim 5n \log_2 n$  has been brought to practice in 2016 in our first work [KS16] as well as in concurrent and independent related work by Lipmaa et al. [LMS16]. Both papers explicitly describe the algorithm for programming Valiant’s universal circuit to compute a given function  $f$ . Moreover, in [KS16], we show the practicality of UCs by providing an open-source implementation of Valiant’s recursive UC construction that uses two substructures at each step of the recursion. Furthermore, we provide an open-source implementation of UC-based PFE (cf. [KS16, Section 4.1]) based on the well-established passively secure SFE framework ABY [DSZ15].

In our second paper on UCs [GKS17], we build on our findings from [KS16] and continue to improve UCs by modularizing our method for defining the program bits in the universal circuit and by applying it to Valiant’s more complicated construction with four substructures at each recursion step and size  $\sim 4.75n \log_2 n$  [Val76]. Furthermore, we provide a hybrid construction (cf. [GKS17, Section 4.2]) that combines both of Valiant’s methods such that it chooses the construction with smaller size at each step of the recursion to achieve the smallest concrete UC to date. We also show that a universal circuit with three substructures is not worth investigating (cf. [GKS17, Section 4.1]) as its size is larger than that of the two constructions presented by Valiant in [Val76].

Our journal paper [AGKS20] extends our results from [KS16; GKS17] by including a recent optimization introduced by Zhao et al. [ZYZL19] which improves the size of UCs (and also our hybrid UC) to  $\sim 4.5n \log_2 n$ . Moreover, we observe that memory consumption becomes the bottleneck of our UC generation and programming implementations. We propose an improved UC generation method (cf. [AGKS20, Section 5.4]) that, instead of utilizing  $\mathcal{O}(n \log n)$  memory by storing the UC at once, uses only  $\mathcal{O}(n)$  memory by handling the

universal circuit layer by layer in a non-trivial manner. Our programming method programs the UC subgraph by subgraph, and therefore, only handles  $\mathcal{O}(n)$  data at once. However, as we store the UC of size  $\mathcal{O}(n \log n)$  (with any additional information until it is necessary) for the generation and programming methods, we require  $\mathcal{O}(n \log n)$  disk storage.

Our works on PFE based on universal circuits have shown that this generic approach is a viable solution for private function evaluation (PFE). We have provided optimizations and improvements, and provide the most efficient open-source UC and UC-based PFE implementation to date at <https://encrypto.de/code/UC>. We have shown that large functions with a million gates can be privately evaluated using universal circuits and the well-established Yao’s garbled circuit protocol [Yao82; Yao86] for SFE within a matter of about a minute in a LAN setting (10 Gbit/s bandwidth and 1 ms RTT) and in about 6 minutes in a WAN setting (100 Mbit/s bandwidth and 100 ms RTT) with about 3.5 GB communication (cf. [AGKS20, Table 7]). Smaller circuit sizes suffice for some simpler use cases mentioned in Section 1.1, such as financial applications or private database management systems (DBMSs), while a million gates may suffice for more complicated use cases as well.

### 2.1.2 PFE using Homomorphic Encryption (HE)

Linear-complexity private function evaluation (PFE) using HE was proposed by Katz and Malka in [KM11]. The authors combine additively homomorphic encryption with garbling techniques [Yao82; Yao86] to achieve PFE. Additively homomorphic encryption allows computation on ciphertexts where the decryption of the encrypted result matches the result of the additive operations as if they had been performed on the plaintext. In garbling techniques, one of the parties, called the garbler, assigns random keys to each wire of a Boolean circuit and using symmetric encryption prepares so-called garbled tables for each gate of the circuit. The other party, called the evaluator, is able to evaluate the Boolean circuit given a set of wire keys for the garbler’s inputs and for his own inputs (which he receives obviously from the garbler without revealing his inputs). In the PFE protocol of [KM11], homomorphic encryption allows one of the parties to garble the circuit obviously without knowing the circuit topology itself. This allows for a PFE protocol with linear complexity  $\mathcal{O}(n)$  in the size of the simulated circuit  $n$ . However, due to the extensive use of HE, this approach has been considered impractical in works on UC-based PFE [KS16; LMS16; GKS17; ZYZL19; AGKS20].

This thesis has contributed to HE-based PFE with the following publication that can be found in Appendix D:

- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: 25. *European Symposium on Research in Computer Security (ESORICS’20)*. Vol. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://encrypto.de/code/linearPFE>. Springer, 2020, pp. 401–420. CORE Rank A. Appendix D.



The linear-complexity PFE protocol of Katz and Malka [KM11] has been brought to practice in our work [HKRS20]. We propose several optimizations and split the protocol in different phases based on if they depend on the private function or the private inputs. Therefore, certain parts of the protocol can be precomputed in many use cases. Our work has shown that PFE using HE is practical when instantiated with state-of-the-art elliptic curve ElGamal and RLWE-based homomorphic encryption. Our most efficient instantiation performs better than our PFE implementation with UCs [AGKS20] in communication for all circuit sizes and in computation on the same platform already starting at a few thousand gates. We have shown that large functions with a million gates can be evaluated privately using HE-based PFE within a matter of 24 seconds in a LAN setting (10 Gbit/s bandwidth and 1 ms RTT) and in less than a minute in a WAN setting (100 Mbit/s bandwidth and 100 ms RTT) with about 330 MB communication (cf. [HKRS20, Tables 1-3]). These runtimes are  $4\text{-}6\times$  smaller than those of UC-based PFE, while the communication is about  $11\times$  smaller.

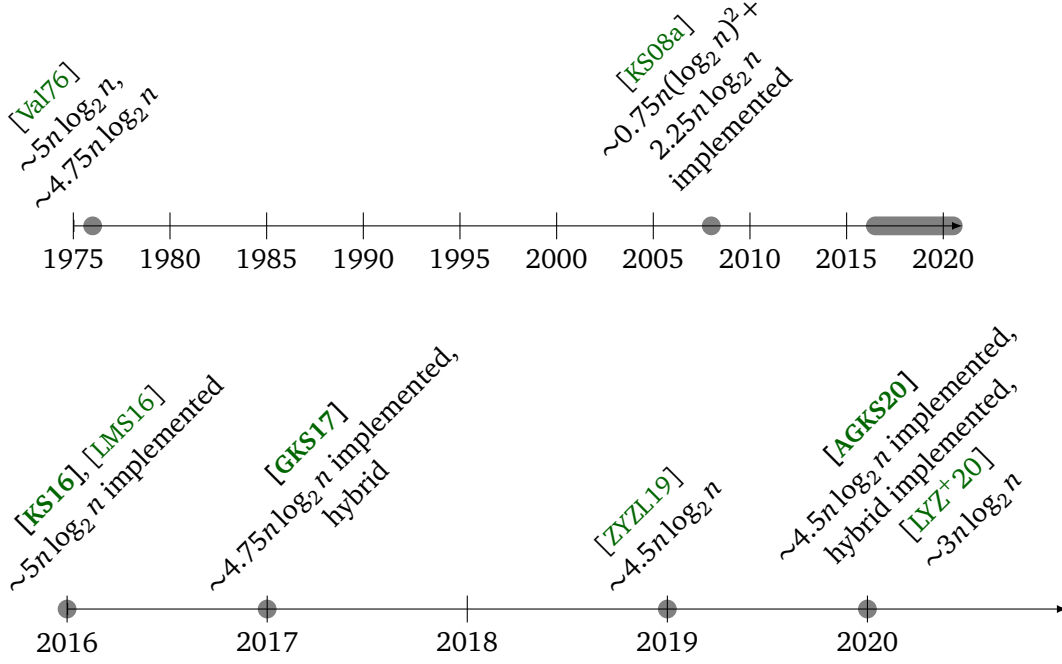
## 2.2 Related Work

Generally, interactive private function evaluation (PFE) approaches utilize either some form of homomorphic encryption (HE) or techniques used in secure function evaluation (SFE) protocols. In this section, we categorize PFE protocols that use the Boolean circuit representation of the private function in four different categories: Secure evaluation of universal circuits (Section 2.2.1), PFE using additive HE (Section 2.2.2), PFE using oblivious transfers (OTs) (Section 2.2.3), and PFE using a trusted execution environment (TEE) (Section 2.2.4). Most of them are secure against semi-honest adversaries, i.e., adversaries that follow the protocol but try to learn secrets during the protocol execution. Some protocols have been extended to malicious adversaries who can actively deviate from the protocol execution to learn private data. We describe these four categories and position our work in the first category of PFE of Boolean circuits using SFE of UCs. Moreover, we show how our work has inspired a new line of research on PFE based on the secure evaluation of universal circuits (UCs) as can be observed on the timeline we depict in Figure 2.1. In the end of this section, we provide a brief comparison (Section 2.2.5).

### 2.2.1 PFE using Secure Function Evaluation (SFE) of Universal Circuits (UCs)

In 1976, Valiant [Val76] proposed the notion of a universal circuit (UC) that can be programmed to compute any Boolean circuit up to a given size  $n$ . Moreover, he introduced two constructions with sizes  $\sim 5n \log_2 n$  and  $\sim 4.75n \log_2 n$  that utilize two and four substructures in their recursion, respectively. Universal circuits were shown to be at least of size  $\Omega(n \log n)$  [Val76; Weg87], i.e., the only improvement left was on the constant factor. Valiant’s construction has then been considered to be mostly a proof of existence of a universal circuit, whereas details needed for the practical realization, e.g., the exact method for deriving the program for the UC were left open. PFE can be reduced to secure function evaluation (SFE) by securely evaluating a UC [AF90; SYY99; Pin02] that is programmed

by  $P_1$  to evaluate function  $f$  represented by circuit  $C$  on  $P_2$ 's input  $x$ .  $P_1$  provides the program bits  $p$  for the UC and  $P_2$  provides his private input  $x$  into an SFE protocol that computes  $UC(x, p) = C(x)$ . In 2008, Kolesnikov and Schneider [KS08a; Sch08] proposed and implemented a novel UC construction with slightly worse asymptotic complexity  $\mathcal{O}(n \log^2 n)$  that was more modular. Moreover, they provided the first PFE implementation based on the Fairplay SFE framework [MNPS04]. The UC construction from [KS08a] was generalized for circuits with gates with more than two inputs in [SS08]. Our work has focused on showing the practicality of Valiant's universal circuit construction from [Val76] that has been abandoned for four decades as can be seen on the timeline depicted in Figure 2.1. In 2016, in our paper [KS16] we have shown that Valiant's UC construction with size  $\sim 5n \log_2 n$  is indeed efficient, described the details of the programming algorithm and provided an open-source implementation of the UC. We have shown in [KS16] that Valiant's UC results in smaller UCs than the asymptotically larger construction of [KS08a] already from a few hundred gates on, and provided the first asymptotically optimal UC-based PFE implementation. In concurrent and independent related work, Lipmaa et al. [LMS16] also showed the practicality of UCs by describing the programming method of the same construction of Valiant [Val76] and decreasing the total number of gates of the UC. However, the number of AND gates in their optimized version remains the same, and therefore, their improvement does not affect PFE with universal circuits where circuits are represented by only AND and XOR gates since XOR gates can be evaluated for free [KS08b]. In [GKS17], we described a more modular approach for generating and programming the UC and extended our open-source implementation with Valiant's method of size  $\sim 4.75n \log_2 n$  [Val76]. Moreover, we proposed a concretely more efficient hybrid UC that combines both constructions to achieve the best concrete UC sizes. In 2019, Zhao et al. [ZYZL19] proposed an improvement for Valiant's UCs by replacing the main block of the construction with size  $\sim 4.75n \log_2 n$  with a smaller block found using exhaustive search over all possible blocks. Their construction results in a UC of size  $\sim 4.5n \log_2 n$ . The authors also provide a lower bound of  $\sim 3.64n \log_2 n$  on Valiant's UC construction [ZYZL19]. We include this construction in our open-source implementation and hybrid UCs in [AGKS20]. Moreover, we propose a scalable UC generation algorithm that does not handle the whole UC of size  $\mathcal{O}(n \log n)$  in memory at once, but handles it layer by layer such that it uses only  $\mathcal{O}(n)$  memory at every step of the execution. This requires careful handling of necessary information for the next layers which are stored using the file system and opened only when necessary. Naturally, as the UC has size  $\mathcal{O}(n \log n)$ , we require  $\mathcal{O}(n \log n)$  disk space for storing the UC and the additional information for the next layers, which are deleted once not required anymore. This improvement allows for generating larger UCs within a given memory constraint. We also show how to program UCs such that at each step of the execution only  $\mathcal{O}(n)$  memory is utilized subgraph by subgraph. Implementing this step, however, is left for future work. The latest improvement on UCs was proposed by Liu et al. [LYZ<sup>+</sup>20], where the authors deviate from the constraints of Valiant's UC framework and modify it such that the resulting UC now has only size  $\sim 3n \log_2 n$  (hence circumventing their lower bound on Valiant's UC construction from [ZYZL19]). They also provide a lower bound of  $\sim 2.95n \log_2 n$  on their construction methodology which almost matches the UC size they achieve. Their construction is simple and more compact, but has not yet been implemented.



**Figure 2.1:** Timeline of research on universal circuits (UCs) and the corresponding UC sizes for private circuits with  $n$  gates.

### 2.2.2 PFE using Homomorphic Encryption (HE)

In 2011, Katz and Malka [KM11] proposed a PFE protocol with linear complexity  $\mathcal{O}(n)$  in the size of the private function  $n$ . The authors use additively homomorphic encryption (HE) to enable the oblivious garbling of the circuit by the party to whom the function should remain private. Mohassel and Sadeghian also include a linear-complexity PFE variant in their framework in [MS13], which is as efficient as the protocol presented by Katz and Malka. Mohassel et al. [MSS14] extend the linear-complexity protocol from [KM11; MS13] to security against malicious adversaries with zero-knowledge proofs. Recently, Biçer et al. [BBKL20] proposed a linear-complexity protocol that becomes more efficient than the protocol of Katz and Malka [KM11] when evaluated multiple times, i.e., it requires lower computation and communication from the second protocol run on, assuming that the same circuit is evaluated privately multiple times. Until recently, PFE using HE has been considered impractical due to its extensive use of homomorphic encryption. We have optimized and brought the protocol of [KM11] into practice in [HKRS20] using state-of-the-art HE schemes such as elliptic curve (EC) ElGamal [Gam85], the Brakerski/Fan-Vercauteren (BFV) scheme [FV12], and the cryptosystem by Damgård-Jurik-Nielsen (DJN) [DJN10], where we provide an implementation of linear-complexity PFE using HE and show that it outperforms PFE using SFE of UCs already for private circuits of a few thousand gates.

### 2.2.3 PFE using Oblivious Transfers (OTs)

Mohassel and Sadeghian [MS13] present a framework for PFE with an instantiation using the oblivious evaluation of a so-called switching network of size  $\mathcal{O}(n \log n)$  in the size of the private circuit  $n$ . These switching networks are obviously evaluated using OTs and hide the topology of the circuit by hiding the mapping that describes how input wires and outgoing wires from gates are connected with incoming wires of subsequent gates and outputs. The well-known half-gates optimization [ZRE15] of Yao’s garbled circuit protocol [Yao82; Yao86] was included in this instantiation by Bingöl et al. [BBKL19] that reduces the number of oblivious transfers by about half.

### 2.2.4 PFE using a Trusted Execution Environment (TEE)

We propose PFE with a trusted execution environment (TEE) in [FKSW19] and provide a prototype implementation using Intel SGX. Here, the trust assumptions are different than those of the other interactive approaches as the TEE is assumed to provide a secure enclave in which the protocol is executed, and PFE is achieved by evaluating a UC within the secure enclave. The function is represented as a universal circuit to avoid side-channel attacks (e.g., [XCP15; BMD<sup>+</sup>17; BWK<sup>+</sup>17; GESM17; HCP17; MIE17; SWG<sup>+</sup>17; SLKP17; WCP<sup>+</sup>17; BMW<sup>+</sup>18; LSG<sup>+</sup>18; MES18; KHF<sup>+</sup>19]), i.e., the program bits are sent to the enclave by the function holder, which then program the UC to compute the private function without revealing the topology or the gates’ functionality to the other party. Therefore, the computation and communication complexity of this solution is that of the size of the UC, i.e.,  $\mathcal{O}(n \log n)$  in the size of the private circuit  $n$ , but the computation is performed within the enclave in a non-interactive manner (after the inputs are sent to the enclave).

### 2.2.5 Comparison

In this section, we compare the computation and communication complexities of all approaches for PFE with full function privacy and refer to our original publications [KS16; GKS17; FKSW19; AGKS20; HKRS20] for details on concrete comparison. We give a brief overview in Table 2.1 for those protocols that have complexity  $\mathcal{O}(n \log n)$  in the size  $n$  of the private circuit, and note that the linear-complexity approaches [KM11; MS13; BBKL20] perform best already from a few thousands of gates as we have shown in [HKRS20]. We note that the TEE-based protocol relies on a different trust model, i.e., it additionally requires trust in a trusted execution environment (TEE).

**Computation.** PFE based on the secure evaluation of universal circuits has been the first PFE approach implemented in [KS08a; KS16; GKS17; AGKS20]. Its most efficient asymptotic computation and communication complexity is  $\Theta(n \log n)$ , concretely it requires four symmetric-key operations for the garbler per AND gate of the universal circuit using Yao’s garbled circuit protocol [AGKS20]. As opposed to this, PFE with oblivious transfers (OTs) [MS13]

	TEE-based PFE	UC-based PFE	OT-based PFE
References	[FKSW19]	[KS16; LMS16; GKS17; ZYZL19; AGKS20]	[MS13; BBKL19]
Section	Section 2.2.4	Section 2.2.1	Section 2.2.3
Computation	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
Communication	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

**Table 2.1:** Comparison of the concrete communication and computation of passively secure private function evaluation approaches with  $\mathcal{O}(n \log n)$  asymptotic complexity. For large enough values of the private circuit size  $n$ , the HE-based PFE protocols [KM11; MS13; BBKL20] with linear complexity  $\mathcal{O}(n)$  (cf. Section 2.2.2) have the lowest computation and communication [HKRS20].

requires about half of the computation of approaches using UCs [AGKS20]. The PFE approach based on evaluating a UC in a TEE also requires  $\mathcal{O}(n \log n)$  computation within the TEE [FKSW19]. Its instantiation with Intel SGX is generally faster in a WAN network setting due to smaller amount of communication [FKSW19] that consists only of the function holder sending the program bits as his inputs to the enclave. This solution, however, relies on additional trust assumptions in the TEE, whereas protocols based on OTs or UCs do not rely on any trusted entity. The protocols based on additively homomorphic encryption (HE) [KM11; BBKL19] require  $\mathcal{O}(n)$  additively homomorphic operations, which, however, are computationally heavier than the symmetric-key operations used by the nonlinear methods. We explored when these linear protocols start outperforming the nonlinear methods based on OTs and UCs in [HKRS20], and found that surprisingly, linear-complexity PFE implemented with the elliptic curve ElGamal HE scheme is more efficient already from a few thousand gates on.

**Communication.** PFE via SFE of UCs has communication complexity  $\mathcal{O}(n \log n)$  as both the universal circuit and the program bits have size  $\mathcal{O}(n \log n)$ . The PFE protocol with OTs from [MS13] requires concretely more communication [GKS17; AGKS20]. Even the improvement from [BBKL19] requires at least twice the communication compared to PFE via SFE of UCs. As communication is considered the bottleneck in protocols where two or more parties collaboratively compute the result in SFE, we note that in practice, the gap in communication between UC-based PFE and OT-based PFE would possibly overcome the gap in computation, especially when using realistic network settings with low bandwidth and high latency. Evaluating the UC in a trusted execution environment (TEE) also requires to send  $\mathcal{O}(n \log n)$  bits since the program bits defining the private function have to be sent to the TEE in this protocol [FKSW19]. The only PFE protocols with linear  $\mathcal{O}(n)$  communication complexity are those based on homomorphic encryption (HE) [KM11; BBKL20]. We investigated the concrete performance of these methods in [HKRS20] and have shown that this asymptotic

difference in communication and the efficiency of recent HE implementations makes it the most efficient approach for PFE to date.

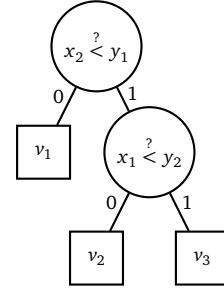
**Adaptability.** PFE via SFE of UCs can easily be used in any secure function evaluation protocol and implementation. Its security follows from that of the SFE protocol that is used to evaluate the UC, i.e., it can be made secure also against malicious adversaries. Moreover, any recent optimization in SFE will benefit this approach as well. Furthermore, outsourcing UC-based PFE to two or multiple servers is directly possible with outsourced SFE [KR11]. Non-interactive secure computation [AMPR14] allows for non-interactive PFE [LMS16], where the first message of the function holder encodes the program bits of the UC that can be evaluated on multiple inputs of the other party, while a second message is required to decode the output. With UC-based PFE, it is directly possible to evaluate public and private circuit parts and allow for a more modular design for a given application as we have shown in [GKSS19]. All other approaches such as the HE-based or OT-based PFE require specific implementation efforts and integration of the protocol for these scenarios.

### 3 Private Function Evaluation of Decision Trees

For privacy-preserving classification such as medical or software diagnosis [BFK<sup>+</sup>09; TASF09; BFL<sup>+</sup>11] mentioned in Section 1.1, decision trees or branching programs (binary decision diagrams) provide a compact representation of the private function. Random forests (sets of smaller decision trees trained on different random subsets of features) are considered to be among the most accurate classification models in machine learning [DCBA14]. Some functions (e.g., multiplication), however, have an exponential-size decision tree representation [Bry91; Woe05; Sch11] and therefore, this representation is not generic and cannot be used in every use case.

In this chapter, we look into approaches for private function evaluation (PFE) where the function is represented as a decision tree, i.e., a binary tree with internal nodes called decision nodes where an input value  $x_i$  is compared to a threshold  $y_j$ , and leaf nodes are called classification nodes where a classification value  $v_k$  is stored as shown in Figure 3.1. Evaluating a decision tree  $T$  on input feature vector  $\mathbf{x}$  means evaluating the comparisons at each decision node of a path, the result of which defines which branch is taken next. When a leaf node is reached, the corresponding classification value is output. The depth of the decision tree is the length of the longest path between the root and any leaf. Here, the challenges for PFE include hiding which input feature from the input feature vector  $\mathbf{x}$  is compared at which decision node, the threshold at each decision node, and the concrete topology of the decision tree (such as the length of individual paths). Generally, the size of the decision tree (i.e., a maximum number of decision nodes and potentially the depth of the tree) is considered to be public in most PFE protocols.

Solutions for privately evaluating decision trees exist and utilize, similarly to techniques for generic PFE, additively homomorphic encryption (HE) or garbling techniques similar to Yao’s garbled circuit protocol for secure two-party computation [Yao82; Yao86]. Additively homomorphic encryption allows for an operation on two ciphertexts that is reflected as an addition of the plaintexts on the decryption of the result. Garbling techniques use (mostly) symmetric-key operations to allow two parties to obliviously compute a Boolean circuit in a constant number of rounds. Techniques based on HE generally require less communication



**Figure 3.1:** Decision tree with  $m = 2$  decision nodes and  $d = 2$  depth.



but have high computation complexity, whereas garbling techniques require more communication but utilize only efficient symmetric-key primitives. In this chapter, we describe our contributions to finding the most efficient protocols combining these two approaches in Section 3.1 and position our work in the literature in Section 3.2.

### 3.1 Our Contributions

Various papers have proposed solutions for privately evaluating decision trees, some based solely on homomorphic encryption (HE) (e.g., [WFNL16; TMZC17]), some solely on garbling techniques (e.g., [BPSW07; BFK<sup>+</sup>09]), and some utilize both techniques (e.g., [BPSW07; BFK<sup>+</sup>09]).

This thesis has contributed to private decision tree evaluation with the following systematization of knowledge publication that can be found in Appendix E:

- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2019.2* (2019). De Gruyter Open. Online version: <https://ia.cr/2018/1099>. Code: <https://crypto.de/code/PDTE>, pp. 187–208. CORE Rank B. Appendix E.

In our work [KNL<sup>+</sup>19], we take a closer look at existing constant-round protocols for private decision tree evaluation and identify three sub-protocols they are composed of: 1) *private selection of input features* that are to be compared with at each of the decision nodes, 2) *private comparison* that takes place at the decision nodes, and 3) *private evaluation of the path* corresponding to the given input in the decision tree. We analyze the state-of-the-art techniques for these sub-protocols that use either homomorphic encryption (H) or garbling techniques (G), and show that they can be combined efficiently and result in six different protocols as shown in Table 3.1, where we denote with ABC the protocol that uses A for the private selection of input features, B for the private comparisons at the decision nodes, and C for the private evaluation of the path such that A, B, and C stand for either homomorphic encryption (H) or garbling techniques (G). Three of these protocols have been proposed in related work (HGG [BPSW07; BFK<sup>+</sup>09], GGG [BFK<sup>+</sup>09] and HHH [TMZC17]), and three of the resulting combinations are discovered by our work (HHG, HGH and GGH) (cf. [KNL<sup>+</sup>19, Section 4.1]).

Some protocols require introducing dummy decision nodes in order to pad all paths to the same length, which is necessary to hide the length of any evaluated path. In this scenario, the number of dummy nodes is  $\mathcal{O}(m^2)$  in the original number of decision nodes  $m$ , which, for decision trees with realistic sizes, is considerably smaller than padding the decision tree to a full tree (cf. [KNL<sup>+</sup>19, Figure 12]), i.e., having  $2^d$  decision nodes at depth  $d$ . For instance, for the UCI dataset diabetes (with 10 input features, depth 28 and 393 decision nodes), the number of decision nodes after depth-padding is 6 432 [KNL<sup>+</sup>19], whereas padding it to a full tree results in a tree with  $2^{28}$ , i.e., over 260 million, decision nodes. We adapt



Protocol	Selection	Comparison	Path evaluation	Depth padding needed	Online rounds
[BPSW07]	H	G	G	no	4
[WFNL16]	H	H	H	no	6
HGG [BFK <sup>+</sup> 09]	H	G	G	yes	4
GGG [BFK <sup>+</sup> 09]	G	G	G	yes	2
HHH [TMZC17]	H	H	H	no	4
HHG [KNL <sup>+</sup> 19]	H	H	G	yes	4
HGH [KNL <sup>+</sup> 19]	H	G	H	no	6
GGH [KNL <sup>+</sup> 19]	G	G	H	no	4

**Table 3.1:** Constant-round protocols composed of the presented sub-protocols for selection, comparison and path evaluation based on homomorphic encryption (H) and garbling techniques (G). We depict if a protocol requires dummy decision nodes to be introduced in order to pad each path to the maximum depth of the decision tree and we show the number of rounds of each protocol.

all protocols into an offline-online setting, i.e., precompute operations independent of the input features in an offline phase to allow for an efficient input-dependent online phase. We experimentally evaluate the performance of the resulting private decision tree evaluation protocols and study the tradeoff between runtime and communication on real-world datasets cf. [KNL<sup>+</sup>19, Section 6]. In a LAN setting (1 Gbit/s bandwidth and 0.5 ms RTT), our newly identified hybrid protocols (HGH and GGH) that use both homomorphic encryption and garbling techniques in a novel combination achieve the fastest runtimes for large datasets while requiring reasonably low communication. For instance, for the above mentioned diabetes dataset, our GGH protocol requires 9.3 MB communication, whereas the lowest communication of 1.6 MB is achieved by the HHH protocol of [TMZC17], which, in turn is more than an order of magnitude slower than GGH that runs in 0.2 s. Moreover, most communication for GGH and HGH can be precomputed offline. Our newly identified hybrid protocols (HGH and GGH) that use both homomorphic encryption and garbling techniques in a novel combination achieve better tradeoffs than state-of-the-art HE-based [WFNL16; TMZC17] or garbling-based [BPSW07; BFK<sup>+</sup>09; BFL<sup>+</sup>11] protocols. Based on our findings, we provide recommendations for usage scenarios (cf. [KNL<sup>+</sup>19, Section 6]) for the most efficient protocols.

### 3.2 Related Work

Recent years have brought tremendous improvements in the field of private decision tree evaluation. First, we group related works into three main categories: Those that utilize homomorphic encryption and/or garbling techniques (Section 3.2.1), and those that utilize some

alternative approaches such as secret sharing (Section 3.2.2) and generic PFE (Section 3.2.3). Within the first category, we position our work in the context of constant-round protocols, and describe related works that are non-interactive as well as where the number of rounds depends on the depth of the decision tree. Then, we briefly compare the different approaches (Section 3.2.4).

#### 3.2.1 Private Decision Tree Evaluation in the Client-Server Setting

We discuss related work on private decision tree evaluation protocols where a server holding its decision tree and a client holding its feature vector perform the privacy-preserving classification themselves. We describe these protocols, grouped together based on their number of rounds, which use different tools in their building blocks as shown in Table 3.2.

**Constant round protocols.** Private function evaluation (PFE) of decision trees was firstly considered by Brickell et al. [BPSW07] for remote diagnostic programs. The authors propose a protocol with mostly garbling techniques, but use homomorphic encryption for the input selection (i.e., protocol HGG). This has been improved and generalized in [BFK<sup>+</sup>09; BFL<sup>+</sup>11], and also a protocol based on mainly symmetric-key operations using Yao’s garbled circuit protocol [Yao82; Yao86] was proposed (i.e., protocol GGG). These approaches are secure against semi-honest adversaries, i.e., adversaries who follow the protocol execution. The solution of [BFK<sup>+</sup>09; BFL<sup>+</sup>11] based only on Yao’s garbled circuit protocol can be extended to protect against a malicious client (the party holding the input to the decision tree) by using oblivious transfer (OT) extension with security against malicious clients [ALSZ15; KOS15; ALSZ17] which is only slightly less efficient than its passively secure variant. Ishai and Paskin [IP07] presented a protocol based on homomorphic encryption for privately evaluating branching programs (a generalization of decision trees) where the server evaluates his program on the client’s encrypted input. Bost et al. [BPTG15] presented a protocol based on leveled fully homomorphic encryption (FHE) where the decision tree is expressed and evaluated as a multivariate polynomial, whose constants are the classification labels and variables represent comparison results at the decision nodes. After the private comparisons, the server homomorphically evaluates the polynomial and returns the result. Wu et al. [WFNL16] proposed a passively secure protocol relying solely on additively homomorphic encryption (HE), and extend their protocol to security against malicious clients. Their protocol allows the server to learn the comparison result encrypted with the client’s public key, and the client to learn the index of the classification label after the server evaluates the tree under encryption. Then, the client learns the result using oblivious transfer (OT). Tai et al. [TMZC17] improved the path evaluation method of this protocol (i.e., protocol HHH) such that they assign costs to each edge, i.e., the left edge of a node has cost  $b$  and the right has cost  $1 - b$ , where  $b$  is the comparison result bit. Then all paths have costs based on the edges along them, and the path with cost zero leads to the classification result. With their improvement, the authors of [TMZC17] eliminate the exponential dependency on the depth of the tree  $d$ , i.e., instead of  $2^d$  decision nodes in [WFNL16], they require only  $m$  decision nodes in their protocol to fully hide the topology of the decision tree. This improvement is significant when large

decision trees are considered that usually are not very dense (see our example decision trees trained on the UCI datasets [Lic18] in [KNL<sup>+</sup>19, Table 3], where all our large trees with more than a hundred decision nodes have less than 0.5% of the decision nodes of a full tree with the same depth). For decision trees with  $m \sim 2^d$ , the protocol of Wu et al. [WFNL16] would perform better, however, for more realistic trees with  $m \leq 2^{d-2}$ , the communication of Tai et al.’s method [TMZC17] is at most half of that of [WFNL16], while the computation is at most the same. De Cock et al. [CDH<sup>+</sup>19] propose a secret sharing-based solution for evaluating private decision trees using the so-called commodity-based model with  $\mathcal{O}(\log t)$  rounds where  $t$  is the bitlength of the inputs that is set to be a constant, usually  $t = 32$  or  $t = 64$ . In this model, correlated randomness is distributed by a trusted party to the computing parties or pre-computed during the offline phase, which are then used in the online phase.

In our work [KNL<sup>+</sup>19], we consider constant-round private decision tree evaluation protocols allow features and inputs to be of any bitlength. We identify the most efficient existing protocols HGG [BPSW07; BFK<sup>+</sup>09], GGG [BFK<sup>+</sup>09], and HHH [TMZC17], and find three novel combinations of their sub-protocols, i.e., HHG, HGH and GGH [KNL<sup>+</sup>19]. HHG is not competitive with other approaches, but the novel hybrid protocols HGH and GGH, combining techniques both from [TMZC17] and [BPSW07; BFK<sup>+</sup>09], and relying on garbled circuits for comparison instead of the DGK comparison protocol [DGK07] result in reasonable performance tradeoffs, and provide the best solution in certain usage scenarios [KNL<sup>+</sup>19].

Most existing constant-round protocols [BPTG15; WFNL16; TMZC17] use a HE-based comparison protocol with the DGK encryption scheme [DGK07; DGK08; DGK09]. Recently, Xue et al. [XLH<sup>+</sup>20] presented an improved comparison protocol based on additively homomorphic encryption and secret sharing that allows the parties to compare their inputs privately as a whole instead of doing it in a bit-by-bit manner as in [DGK07; DGK08; DGK09]. Then they use the path evaluation of Tai et al. [TMZC17] and achieve a private decision tree evaluation protocol that requires a factor  $t$  less homomorphic operations for the comparisons than previous works where  $t$  is the constant bitlength of the inputs and thresholds.

**Non-constant round protocols.** The constant-round protocols based solely on HE [BPTG15; WFNL16; TMZC17] use the DGK comparison protocol [DGK07]. Joye and Salehi [JS18] propose an optimization of the DGK protocol that improves its communication by a factor two when comparing two values. However, the DGK comparison allows for including the input feature selection step without additional overhead, while this optimization incurs additional overhead when more values are compared, such as in case of private decision tree evaluation. Moreover, their protocol has  $\mathcal{O}(d)$  rounds [JS18], where  $d$  denotes the depth of the decision tree. Tueno et al. [TKK19] represent the decision tree as an array, and execute  $d$  comparisons. These comparisons are performed using garbled circuits that result in the secret-shared index of the next node. Then they perform oblivious array indexing which allows the parties to obliviously select the next nodes. For this, they utilize either garbled circuits, oblivious transfer or oblivious RAM (ORAM), the latter of which results in a protocol with sub-linear complexity

	Protocol	Tools	Rounds	Communication	Comparisons
Constant round	[BPSW07]	HE, GC	5	$\mathcal{O}(2^d)$	$m$
	HGG [BFK <sup>+</sup> 09]	HE, GC	4	$\mathcal{O}(m^2)$	$m$
	GGG [BFK <sup>+</sup> 09]	GC	4	$\mathcal{O}(m^2 \log(m^2))$	$m$
	[BPTG15]	FHE, HE	$\geq 6$	$\mathcal{O}(m)$	$m$
	[WFNL16]	HE, OT	6	$\mathcal{O}(m)$	$m$
	HHH [TMZC17]	HE	4	$\mathcal{O}(m)$	$m$
	HGH [KNL <sup>+</sup> 19]	HE, GC	6	$\mathcal{O}(m)$	$m$
	GGH [KNL <sup>+</sup> 19]	HE, GC	6	$\mathcal{O}(m \log m)$	$m$
	[CDH <sup>+</sup> 19]	SS	9	$\mathcal{O}(2^d)$	$m$
	[XLH <sup>+</sup> 20]	HE, SS	4	$\mathcal{O}(m)$	$m$
Non-constant round	[TKK19]	GC, OT	$4d$	$\mathcal{O}(2^d)$	$d$
	[TKK19]	ORAM	$d^2 + 3d$	$\mathcal{O}(d^4)$	$d$
	[TKK19]	ODS	$4d$	$\mathcal{O}(d^3)$	$d$
	[TKK19]	ORAM, FSS	$4d$	$\mathcal{O}(d^2)$	$d$
	[JS18]	HE, OT	$2d$	$\mathcal{O}(2^d)$	$d$
	GGG [BFK <sup>+</sup> 09]	GC	$2d$	$\mathcal{O}(m^2 \log(m^2))$	$d$
Non-interactive	[LZS18]	FHE, HE	1	$\mathcal{O}(m)$	$m$
	[TBK20] binary	FHE, HE	1	$\mathcal{O}(1)$ or $\mathcal{O}(d)$	$m$
	[TBK20] integer	FHE, HE	1	$\mathcal{O}(2^d/s)$ or $\mathcal{O}(d)$	$m$
	[ALR <sup>+</sup> 19]	FHE	1	$\mathcal{O}(1)$	$m$

**Table 3.2:** Summary of private decision tree evaluation protocols in the client-server setting. Table modified and extended from [TKK19; TBK20], assuming that the number of input features is  $\mathcal{O}(m)$  in the number of decision nodes  $m$ .  $d$  denotes the depth of the decision tree. For the underlying tools, we use the following notation: fully homomorphic encryption (FHE), homomorphic encryption (HE), garbled circuit (GC), oblivious transfer (OT), secret sharing (SS), function secret sharing (FSS), oblivious random access memory (ORAM), oblivious data structure (ODS).

in the size of the decision tree. Their protocols require the same number of comparisons as the protocol of [JS18], but require at least  $\mathcal{O}(d)$  rounds of communication [TKK19, Tab. 2].

**Non-interactive protocols.** An alternative to the DGK comparison protocol [DGK07; DGK08; DGK09] is the non-interactive comparison protocol presented by Lu et al. [LZS18] that uses fully homomorphic encryption (FHE). The authors implement the protocol of [TMZC17] with their comparison protocol to achieve a non-interactive private decision tree evaluation protocol. Their solution only performs well for small bitlengths of the features and inputs. Tueno et al. [TBK20] further improve non-interactive private decision tree evaluation based on FHE that allows a client to delegate the decision tree evaluation to the server by sending an encrypted input and receiving an encrypted result. Their solution performs well for low-depth trees but its runtime grows exponentially in the depth, and therefore, is not

applicable for deep decision trees. The recent work of Akavia et al. [ALR<sup>+</sup>19] presents a non-interactive solution with FHE that is secure against malicious adversaries and allows for sublinear client complexity in the size of the decision tree.

#### 3.2.2 Private Decision Tree Evaluation in the Outsourcing Setting

Aloufi et al. [AHC19] propose outsourced evaluation of random forests of multiple model owners and use multi-key HE. The model owners send encrypted decision trees with joint keys and the client sends its encrypted input with its key to the cloud, who evaluates the trees securely and aggregates the results. In the end of the protocol execution, the result is decrypted using all HE keys. Zheng et al. [ZDW19] proposed an approach for outsourcing private decision tree evaluation based on additive secret sharing and provide a tailored protocol that allows the parties to stay offline after providing their inputs to the computation. Liu et al. [LSC<sup>+</sup>19; LCL<sup>+</sup>20; LSZ<sup>+</sup>20] propose a solution based on additively homomorphic encryption and secret sharing that allows the client to outsource the private decision tree evaluation to two servers in the cloud. In their solution, the client does not learn anything about the decision tree while the servers learn nothing about the client's input as long as they do not collude with each other.

#### 3.2.3 Private Decision Tree Evaluation with Private Function Evaluation of Boolean Circuits

Generic private function evaluation (PFE) of Boolean circuits (cf. Chapter 2) can also be used to solve private decision tree evaluation, in which case the decision tree has to be transformed into a Boolean circuit representation. This, however, implies a very large overhead for a decision tree with bitlength  $t$  of the features and  $m$  decision nodes since for input selection, a selection network of  $\mathcal{O}(tm \log m)$  gates described in [KNL<sup>+</sup>19, Section 3.1] is utilized. Then, PFE based on universal circuits (UCs) and OTs [Val76; KS16; LMS16; GKS17; ZYZL19; AGKS20; LYZ<sup>+</sup>20] have complexity  $\mathcal{O}((tm \log m) \log(tm \log m))$ , and linear-complexity HE-based PFE protocols [KM11; MS13; MSS14; HKRS20] have complexity  $\mathcal{O}(tm \log m)$ , which are significantly worse than protocols designed specifically for decision trees shown in Table 3.2.

#### 3.2.4 Comparison

Protocols designed specifically for private decision tree evaluation can be grouped into two categories: The first solution relies on homomorphic encryption and/or garbling techniques (cf. Section 3.2.1), where the client holding an input feature vector and a server holding a decision tree compute the result interactively in a privacy-preserving manner. The second is based on secret sharing (cf. Section 3.2.2) where the client outsources the decision tree computation to two or more non-colluding servers. The trust model of the two approaches differ significantly, as secret sharing-based approaches require two non-colluding computing

servers, a stronger assumption the client has to make than in case of the approaches based on HE and/or garbling techniques, where the server and the client themselves perform the computation. Secure outsourcing of the computation, however, can be more efficient, especially when the client has a computationally restricted device.

Protocols where the client and server perform the private decision tree evaluation interactively have gained increasing interest with recent advances of machine learning. We have shown in [KNL<sup>+</sup>19] that for constant-round protocols that are based on homomorphic encryption and garbling techniques, it is beneficial to combine these techniques and design protocols that use both in order to provide the best tradeoff between computation and communication, as garbling techniques are fast but heavy in communication, while homomorphic encryption is slower but requires less communication. Our work provides performance comparison of existing techniques on the same platform [KNL<sup>+</sup>19, Section 6]. The recent work of Xue et al. [XLH<sup>+</sup>20] combines additively homomorphic encryption with secret sharing and achieves an improvement as comparisons are not performed bit by bit anymore.

Considering real-world scenarios with high latency network connection between the client and the server, non-constant round protocols (cf. Section 3.2.1) would provide worse performance in real-world use cases. However, the recently proposed non-interactive private decision tree evaluation protocols of [LZS18; ALR<sup>+</sup>19; TBK20] that rely on fully homomorphic encryption (FHE) may provide the best solution in some real-world applications with high latency between the two parties since the client only needs to send its encrypted input and receive the encrypted output. Though the client is still required to perform a computationally heavy encryption and decryption, the decision tree evaluation is performed solely by the server under encryption.

Using generic private function evaluation (PFE) for privately evaluating decision trees (cf. Section 3.2.3) is inefficient and should only be considered when small decision trees are evaluated and generic PFE is easily applied. Otherwise specific protocols with better complexity should be utilized.

## 4 Conclusion and Future Work

---

In this chapter, we conclude this thesis by summarizing our main contributions to the field of efficient private function evaluation (PFE) in Section 4.1, and by describing possible future work directions in Section 4.2.

### 4.1 Summary

This thesis has made significant contributions towards making PFE efficient and scalable by improving generic PFE based on the secure evaluation of universal circuits (UCs) and based on homomorphic encryption (HE) summarized in Chapter 2, as well as specific PFE of decision trees summarized in Chapter 3. Here, we briefly describe our findings.

**More Efficient Generic Private Function Evaluation.** In [KS16], we have shown that asymptotically optimal universal circuits (UCs) as proposed by Valiant [Val76] can be brought into practice, providing the first generic PFE implementation with UCs of asymptotically optimal complexity  $\mathcal{O}(n \log n)$ . We have described and implemented the algorithm that generates and programs the UC according to the function it should simulate. Our UC design in [GKS17] is modular, and therefore, we could include in [AGKS20] recent improvements both from ourselves and from other research groups [ZYZL19]. We have shown that PFE using UCs is an efficient solution for private function evaluation and provide a publicly available prototype implementation. In [HKRS20], we optimize and implement the linear-complexity PFE protocol of [KM11] using additively homomorphic encryption that was believed to be inefficient. We show that using state-of-the-art HE schemes such as elliptic curve ElGamal encryption and RLWE-based HE, this protocol has lower communication as well as faster runtimes than UC-based PFE for circuits of at least a few thousand gates.

**Hybrid Private Decision Tree Evaluation.** We have studied constant-round protocols for privately evaluating decision trees, a common machine learning classifier. In private decision tree evaluation, the server holding the decision tree and the client holding the input feature would like to evaluate the decision tree and allow the client to learn the result without compromising the privacy of their inputs. Previously, it has mostly been realized with additively homomorphic encryption [WFNL16; TMZC17] or with garbling techniques [BPSW07; BFK<sup>+</sup>09; BFL<sup>+</sup>11]. We have shown in [KNL<sup>+</sup>19] that novel combinations of these techniques resulting in hybrid protocols provide better tradeoffs between computation and communication than state-of-the-art methods.



## 4.2 Future Work

In this section, we describe future work directions for both function representations considered in this thesis, i.e., for private function evaluation (PFE) of Boolean circuits (Section 4.2.1) and of decision trees (Section 4.2.2).

### 4.2.1 PFE of Boolean Circuits

Even though significant progress has been made in generic PFE, it has not yet reached the level of practicality that would be desired for real-world deployment. Here we describe a few directions to explore in future work.

**Approaches based on universal circuits.** Liu et al. [LYZ<sup>+</sup>20] have shown a lower bound of  $\sim 2.95n \log_2 n$  on the size of UCs along with a construction that achieves size  $\sim 3n \log_2 n$ . It is an interesting future work to implement this construction and see if hybrid methods that we proposed in [GKS17] are still sensible or if they become obsolete due to this novel optimization as [LYZ<sup>+</sup>20, Figure 12] suggests. Moreover, currently the only existing implementation of PFE with UCs is our open-source implementation based on the passively secure two-party secure function evaluation (SFE) framework ABY [DSZ15]. Any SFE framework with two or multiple parties could be adapted to evaluate UCs, so achieving malicious security and multi-party PFE is straightforward, but has not yet been implemented. Secret sharing-based approaches are not efficient here as their round complexity depends on the depth of the UC, which is linear in the size of the private function. We have shown in [AGKS20, Figure 15, Table 7] that this heavily impacts the runtime of PFE in realistic network settings. Therefore, multi-party solutions with a constant number of rounds such as the BMR protocol [BMR90; LPSY15; BLO16; BLO17] should be investigated. Deployment of PFE with UCs by companies that provide solutions based on secure function evaluation should be rather straightforward and is an important future work to be done to enhance the level of privacy in use cases where the function is the trade secret of one of the participants.

**Approaches based on homomorphic encryption.** In [HKRS20], we investigate the concrete efficiency of the constant-round HE-based method of Katz and Malka [KM11] since its communication is asymptotically lower than that of UC-based PFE. It was expected that this method outperforms UC-based PFE for large circuits, however, we show that it becomes more efficient already from circuits of a few thousands of gates on when instantiated with elliptic curve ElGamal HE. It would be interesting to see the performance improvement of the reusable protocol of Biçer et al. [BBKL20] for multiple executions. [KM11; BBKL20] provide protocols with passive security, and Mohassel et al. [MSS14] provide an extension for malicious security with zero-knowledge proofs. After having investigated the breaking point between passively secure PFE with UCs and passively secure PFE with HE in [HKRS20, Section 5.2], an interesting next step would be to do the same for efficient maliciously secure implementations.



#### 4.2.2 PFE of Decision Trees

Privately evaluating decision tree classifiers has gained increasing interest with advances in machine learning. Available solutions are promising but different scenarios require different approaches. We consider constant-round protocols and non-interactive protocols based on fully homomorphic encryption (FHE) as most promising, since with realistic network settings (with high latency and/or low bandwidth) and decision tree sizes, protocols with a non-constant number of rounds become highly inefficient.

**Constant-round approaches based on garbling techniques and additive HE.** As future work, it is interesting to explore malicious security for our novel hybrid protocols HGH and GGH [KNL<sup>+</sup>19], as well as for the protocol HHH of [TMZC17], and elaborate on the performance of all efficient protocols in this setting as well. Further improving the different sub-protocols used for private decision tree evaluation is also of interest as shown in [XLH<sup>+</sup>20] where a novel *comparison protocol* is presented. A direct efficiency comparison between this new method and our hybrid protocols HGH and GGH [KNL<sup>+</sup>19] as well as protocol HHH of Tai et al. [TMZC17] would be interesting, as they perform the path evaluation with the same method. However, since the *path evaluation protocol* of [TMZC17] using HE already compares only at each decision node, we do not expect a significant improvement on this sub-protocol. This is due to the fact that any protocol requiring only one comparison at each layer would also require  $\mathcal{O}(d)$  rounds in the depth  $d$  of the tree, since naturally, in order to compute exactly one path obviously, the parties need communication after each decision node on the path.

**Non-interactive approaches based on FHE.** Novel approaches that use FHE for private decision tree evaluation require no interaction during the computation between the input provider and the model holder (except for sending an encrypted input and receiving an encrypted output). Recent works have shown that this approach may become practical by providing prototype implementation [LZS18; ALR<sup>+</sup>19; TBK20], but the concrete gap between these methods relying on FHE and those relying on computationally less expensive primitives that we have considered is still to be explored in real-world network settings. These approaches, if made generic, i.e., applicable for any bitlength and decision tree topology, may perform better than constant-round protocols.

## Bibliography

---

- [AF90] M. ABADI, J. FEIGENBAUM. “**Secure Circuit Evaluation**”. In: *Journal of Cryptology (JoC)* 2.1 (1990). Springer, pp. 1–12.
- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Online version: <https://ia.cr/2019/348>. Code: <https://encrypto.de/code/UC>. Springer, pp. 1216–1271.
- [AHCW19] A. ALOUFI, P. HU, H. W. H. WONG, S. S. M. CHOW. “**Blindfolded Evaluation of Random Forests with Multi-Key Homomorphic Encryption**”. In: (2019). IEEE.
- [ALR<sup>+</sup>19] A. AKAVIA, M. LEIBOVICH, Y. S. RESHEFF, R. RON, M. SHAHAR, M. VALD. “**Privacy-Preserving Decision Tree Training and Prediction against Malicious Server**”. IACR Cryptology ePrint Archive, Report 2019/1282. <https://ia.cr/2019/1282>. 2019.
- [ALSZ15] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. “**More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries**”. In: *Advances in Cryptology – EUROCRYPT’15*. Vol. 9056. LNCS. Springer, 2015, pp. 673–701.
- [ALSZ17] G. ASHAROV, Y. LINDELL, T. SCHNEIDER, M. ZOHNER. “**More Efficient Oblivious Transfer Extensions**”. In: *Journal of Cryptology (JoC)* 30.3 (2017). Springer, pp. 805–858.
- [AM18] B. A. ALAHMADI, I. MARTINOVIC. “**MalClassifier: Malware Family Classification using Network Flow Sequence Behaviour**”. In: *APWG Symposium on Electronic Crime Research (eCrime’18)*. IEEE, 2018, pp. 1–13.
- [AMPR14] A. AFSHAR, P. MOHASSEL, B. PINKAS, B. RIVA. “**Non-Interactive Secure Computation Based on Cut-and-Choose**”. In: *Advances in Cryptology – EUROCRYPT’14*. Vol. 8441. LNCS. Springer, 2014, pp. 387–404.
- [BBKL19] O. BIÇER, M. A. BINGÖL, M. S. KIRAZ, A. LEVI. “**Towards Practical PFE: An Efficient 2-Party Private Function Evaluation Protocol Based on Half Gates**”. In: *The Computer Journal* 62.4 (2019). Oxford University Press, pp. 598–613.
- [BBKL20] O. BIÇER, M. A. BINGÖL, M. S. KIRAZ, A. LEVI. “**Highly Efficient and Re-executable Private Function Evaluation with Linear Complexity**”. In: *IEEE Transactions on Dependable and Secure Computing* (2020). To appear.
- [BDK<sup>+</sup>18] N. BÜSCHER, D. DEMMLER, S. KATZENBEISSER, D. KRETZMER, T. SCHNEIDER. “**HyCC: Compilation of Hybrid Protocols for Practical Secure Computation**”. In: *ACM Conference on Computer and Communications Security (CCS’18)*. ACM, 2018, pp. 847–861.
- [BFK<sup>+</sup>09] M. BARNI, P. FAILLA, V. KOLESNIKOV, R. LAZZERETTI, A.-R. SADEGHI, T. SCHNEIDER. “**Secure Evaluation of Private Linear Branching Programs with Medical Applications**”. In: *European Symposium on Research in Computer Security (ESORICS’09)*. Vol. 5789. LNCS. Springer, 2009, pp. 424–439.

- [BFL<sup>+</sup>11] M. BARNI, P. FAILLA, R. LAZZERETTI, A.-R. SADEGHI, T. SCHNEIDER. “**Privacy-Preserving ECG Classification With Branching Programs and Neural Networks**”. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 6.2 (2011). IEEE, pp. 452–468.
- [BLO16] A. BEN-EFRAIM, Y. LINDELL, E. OMRI. “**Optimizing Semi-Honest Secure Multiparty Computation for the Internet**”. In: *ACM Conference on Computer and Communications Security (CCS’16)*. ACM, 2016, pp. 578–590.
- [BLO17] A. BEN-EFRAIM, Y. LINDELL, E. OMRI. “**Efficient Scalable Constant-Round MPC via Garbled Circuits**”. In: *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Springer, 2017, pp. 471–498.
- [BMD<sup>+</sup>17] F. BRASSER, U. MÜLLER, A. DMITRIENKO, K. KOSTIAINEN, S. CAPKUN, A.-R. SADEGHI. “**Software Grand Exposure: SGX Cache Attacks Are Practical**”. In: *USENIX Workshop on Offensive Technologies (WOOT’17)*. USENIX Association, 2017.
- [BMR90] D. BEAVER, S. MICALI, P. ROGAWAY. “**The Round Complexity of Secure Protocols (Extended Abstract)**”. In: *ACM Symposium on Theory of Computing (STOC’90)*. ACM, 1990, pp. 503–513.
- [BMW<sup>+</sup>18] J. V. BULCK, M. MINKIN, O. WEISSE, D. GENKIN, B. KASIKCI, F. PIESSENS, M. SILBERSTEIN, T. F. WENISCH, Y. YAROM, R. STRACKX. “**Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution**”. In: *USENIX Security Symposium 2018*. USENIX Association, 2018, pp. 991–1008.
- [BPSW07] J. BRICKELL, D. E. PORTER, V. SHMATIKOV, E. WITCHEL. “**Privacy-Preserving Remote Diagnostics**”. In: *ACM Conference on Computer and Communications Security (CCS’07)*. ACM, 2007, pp. 498–507.
- [BPTG15] R. BOST, R. A. POPA, S. TU, S. GOLDWASSER. “**Machine Learning Classification over Encrypted Data**”. In: *Network and Distributed System Security Symposium (NDSS’15)*. The Internet Society, 2015.
- [Bry91] R. E. BRYANT. “**On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication**”. In: *IEEE Transactions on Computers* 40.2 (1991), pp. 205–213.
- [BWK<sup>+</sup>17] J. V. BULCK, N. WEICHBRODT, R. KAPITZA, F. PIESSENS, R. STRACKX. “**Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution**”. In: *USENIX Security Symposium 2017*. USENIX Association, 2017, pp. 1041–1056.
- [CDH<sup>+</sup>19] M. D. COCK, R. DOWSLEY, C. HORST, R. KATTI, A. C. A. NASCIMENTO, W.-S. POON, S. C. TRUEX. “**Efficient and Private Scoring of Decision Trees, Support Vector Machines and Logistic Regression Models based on Pre-Computation**”. In: *IEEE Transactions on Dependable and Secure Computing (TDSC)* 16.2 (2019). IEEE, pp. 217–230.
- [DCBA14] M. F. DELGADO, E. CERNADAS, S. BARRO, D. G. AMORIM. “**Do We Need Hundreds of Classifiers to Solve Real World Classification Problems?**” In: *Journal of Machine Learning Research (JMLR)* 15.1 (2014). Microtome Publishing, pp. 3133–3181.
- [DGK07] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD. “**Efficient and Secure Comparison for On-Line Auctions**”. In: *Australasian Conference on Information Security and Privacy (ACISP’07)*. Vol. 4586. LNCS. Springer, 2007, pp. 416–430.

- [DGK08] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD. **“Homomorphic Encryption and Secure Comparison”**. In: *International Journal of Applied Cryptography (IJACT)* 1.1 (2008). Inderscience, pp. 22–31.
- [DGK09] I. DAMGÅRD, M. GEISLER, M. KRØIGAARD. **“A Correction to ‘Efficient and Secure Comparison for On-Line Auctions’”**. In: *International Journal of Applied Cryptography (IJACT)* 1.4 (2009). Inderscience, pp. 323–324.
- [DJN10] I. DAMGÅRD, M. JURIK, J. B. NIELSEN. **“A generalization of Paillier’s public-key system with applications to electronic voting”**. In: *International Journal of Information Security* 9.6 (2010), pp. 371–385.
- [DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZÖHNER. **“ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation”**. In: *Network and Distributed System Security Symposium (NDSS’15)*. Code: <https://crypto.de/code/ABY>. The Internet Society, 2015.
- [EPI19] T. v. ELSLOO, G. PATRINI, H. IVEY-LAW. **“SEALion: a Framework for Neural Network Inference on Encrypted Data”**. CoRR abs/1904.12840. <http://arxiv.org/abs/1904.12840>. 2019.
- [Eur16] EUROPEAN PARLIAMENT, COUNCIL OF THE EUROPEAN UNION. **“Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation)”**. In: *Official Journal of the European Union* L.119 (2016). European Union, pp. 1–88.
- [FAL06] K. B. FRIKKEN, M. J. ATALLAH, J. LI. **“Attribute-Based Access Control with Hidden Policies and Hidden Credentials”**. In: *IEEE Transactions on Computers* 55.10 (2006). IEEE, pp. 1259–1270.
- [FAZ05] K. B. FRIKKEN, M. J. ATALLAH, C. ZHANG. **“Privacy-Preserving Credit Checking”**. In: *ACM Conference on Electronic Commerce (EC’05)*. ACM, 2005, pp. 147–154.
- [FKSW19] S. FELSEN, Á. KISS, T. SCHNEIDER, C. WEINERT. **“Secure and Private Function Evaluation with Intel SGX”**. In: *ACM Cloud Computing Security Workshop (CCSW’19)*. ACM, 2019, pp. 165–181.
- [FLA06] K. B. FRIKKEN, J. LI, M. J. ATALLAH. **“Trust Negotiation with Hidden Credentials, Hidden Policies, and Policy Cycles”**. In: *Network and Distributed System Security Symposium (NDSS’06)*. The Internet Society, 2006, pp. 157–172.
- [FV12] J. FAN, F. VERCAUTEREN. **“Somewhat Practical Fully Homomorphic Encryption”**. Cryptology ePrint Archive, Report 2012/144. <https://ia.cr/2012/144>. 2012.
- [FVK<sup>+</sup>15] B. FISCH, B. VO, F. KRELL, A. KUMARASUBRAMANIAN, V. KOLESNIKOV, T. MALKIN, S. M. BELLOVIN. **“Malicious-Client Security in Blind Seer: A Scalable Private DBMS”**. In: *IEEE Symposium on Security and Privacy (S&P’15)*. IEEE, 2015, pp. 395–410.
- [Gam85] T. E. GAMAL. **“A public key cryptosystem and a signature scheme based on discrete logarithms”**. In: *IEEE Transactions on Information Theory* 31.4 (1985). IEEE, pp. 469–472.
- [GESM17] J. GÖTZFRIED, M. ECKERT, S. SCHINZEL, T. MÜLLER. **“Cache Attacks on Intel SGX”**. In: *European Workshop on Systems Security (EUROSEC’17)*. ACM, 2017, 2:1–2:6.

- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Full version: <https://ia.cr/2017/798>, Code: <https://encrypto.de/code/UC>. Springer, 2017, pp. 443–470.
- [GKSS19] D. GÜNTHER, Á. KISS, L. SCHEIDEL, T. SCHNEIDER. “**Framework for Semi-Private Function Evaluation with Application to Secure Insurance Rate Calculation**”. In: *ACM Conference on Computer and Communications Security (CCS’19) Poster/Demo*. ACM, 2019, pp. 2541–2543.
- [HCP17] M. HÄHNEL, W. CUI, M. PEINADO. “**High-Resolution Side Channels for Untrusted Operating Systems**”. In: *USENIX Annual Technical Conference (USENIX ATC’17)*. USENIX Association, 2017, pp. 299–312.
- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: *European Symposium on Research in Computer Security (ESORICS’20)*. Vol. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://encrypto.de/code/linearPFE>. Springer, 2020, pp. 401–420.
- [IP07] Y. ISHAI, A. PASKIN. “**Evaluating Branching Programs on Encrypted Data**”. In: *Theory of Cryptography Conference (TCC’07)*. Vol. 4392. LNCS. Springer, 2007, pp. 575–594.
- [JS18] M. JOYE, F. SALEHI. “**Private yet Efficient Decision Tree Evaluation**”. In: *Data and Applications Security and Privacy (DBSec’18)*. Vol. 10980. LNCS. Springer, 2018, pp. 243–259.
- [JVC18] C. JUVEKAR, V. VAIKUNTANATHAN, A. CHANDRAKASAN. “**GAZELLE: A Low Latency Framework for Secure Neural Network Inference**”. In: *USENIX Security Symposium 2018*. USENIX, 2018, pp. 1651–1669.
- [KHF<sup>+</sup>19] P. KOCHER, J. HORN, A. FOGH, D. GENKIN, D. GRUSS, W. HAAS, M. HAMBURG, M. LIPP, S. MANGARD, T. PRESCHER, M. SCHWARZ, Y. YAROM. “**Spectre Attacks: Exploiting Speculative Execution**”. In: *IEEE Symposium on Security and Privacy (S&P’19)*. IEEE, 2019, pp. 1–19.
- [KKW17] W. S. KENNEDY, V. KOLESNIKOV, G. T. WILFONG. “**Overlaying Conditional Circuit Clauses for Secure Computation**”. In: *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Springer, 2017, pp. 499–528.
- [KM11] J. KATZ, L. MALKA. “**Constant-Round Private Function Evaluation with Linear Complexity**”. In: *Advances in Cryptology – ASIACRYPT’11*. Vol. 7073. LNCS. Springer, 2011, pp. 556–571.
- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2019.2* (2019). Online version: <https://ia.cr/2018/1099>. Code: <https://encrypto.de/code/MobilePSI>. De Gruyter Open, pp. 187–208.
- [Kol18] V. KOLESNIKOV. “**Free IF: How to Omit Inactive Branches and Implement S-Universal Garbled Circuit (Almost) for Free**”. In: *Advances in Cryptology – ASIACRYPT’18*. Vol. 11274. LNCS. Springer, 2018, pp. 34–58.
- [KOS15] M. KELLER, E. ORSINI, P. SCHOLL. “**Actively Secure OT Extension with Optimal Overhead**”. In: *Advances in Cryptology – CRYPTO’15*. Vol. 9215. LNCS. Springer, 2015, pp. 724–741.
- [KR11] S. KAMARA, M. RAYKOVA. “**Secure Outsourced Computation in a Multi-Tenant Cloud**”. In: *IBM Workshop on Cryptography and Security in Clouds 2011*. 2011.

- [KS08a] V. KOLESNIKOV, T. SCHNEIDER. “**A Practical Universal Circuit Construction and Secure Evaluation of Private Functions**”. In: *Financial Cryptography and Data Security (FC’08)*. Vol. 5143. LNCS. Springer, 2008, pp. 83–97.
- [KS08b] V. KOLESNIKOV, T. SCHNEIDER. “**Improved Garbled Circuit: Free XOR Gates and Applications**”. In: *International Colloquium on Automata, Languages and Programming (ICALP’08)*. Vol. 5126. LNCS. Springer, 2008, pp. 486–498.
- [KS16] Á. KISS, T. SCHNEIDER. “**Valiant’s Universal Circuit is Practical**”. In: *Advances in Cryptology – EUROCRYPT’16*. Vol. 9665. LNCS. Full version: <https://ia.cr/2016/093>, Code: <https://encrypto.de/code/UC>. Springer, 2016, pp. 699–728.
- [LCL<sup>+</sup>20] L. LIU, R. CHEN, X. LIU, J. SU, L. QIAO. “**Towards Practical Privacy-Preserving Decision Tree Training and Evaluation in the Cloud**”. In: *IEEE Transactions on Information Forensics and Security (TIFS)* 15 (2020). IEEE, pp. 2914–2929.
- [Lic18] M. LICHMAN. “**UCI Machine Learning Repository**”. Accessed: 2018-08-24. 2018.
- [LJLA17] J. LIU, M. JUUTI, Y. LU, N. ASOKAN. “**Oblivious Neural Network Predictions via MiniONN Transformations**”. In: *ACM Conference on Computer and Communications Security (CCS’17)*. ACM, 2017, pp. 619–631.
- [LMS16] H. LIPMAA, P. MOHASSEL, S. S. SADEGHIAN. “**Valiant’s Universal Circuit: Improvements, Implementation, and Applications**”. IACR Cryptology ePrint Archive, Report 2016/017. <https://ia.cr/2016/017>. 2016.
- [LPSY15] Y. LINDELL, B. PINKAS, N. P. SMART, A. YANAI. “**Efficient Constant Round Multi-party Computation Combining BMR and SPDZ**”. In: *Advances in Cryptology - CRYPTO 2015*. Vol. 9216. LNCS. Springer, 2015, pp. 319–338.
- [LSC<sup>+</sup>19] L. LIU, J. SU, R. CHEN, J. CHEN, G. SUN, J. LI. “**Secure and Fast Decision Tree Evaluation on Outsourced Cloud Data**”. In: *Machine Learning for Cyber Security (ML4CS’19)*. Vol. 11806. LNCS. Springer, 2019, pp. 361–377.
- [LSG<sup>+</sup>18] M. LIPP, M. SCHWARZ, D. GRUSS, T. PRESCHER, W. HAAS, A. FOGH, J. HORN, S. MANGARD, P. KOCHER, D. GENKIN, Y. YAROM, M. HAMBURG. “**Meltdown: Reading Kernel Memory from User Space**”. In: *USENIX Security Symposium 2018*. USENIX Association, 2018, pp. 973–990.
- [LSZ<sup>+</sup>20] L. LIU, J. SU, B. ZHAO, Q. WANG, J. CHEN, Y. LUO. “**Towards an Efficient Privacy-Preserving Decision Tree Evaluation Service in the Internet of Things**”. In: *Symmetry* 12.1 (2020). MDPI AG, p. 103.
- [LYZ<sup>+</sup>20] H. LIU, Y. YU, S. ZHAO, J. ZHANG, W. LIU. “**Pushing the Limits of Valiant’s Universal Circuits: Simpler, Tighter and More Compact**”. IACR Cryptology ePrint Archive, Report 2020/161. <https://ia.cr/2020/161>. 2020.
- [LZS18] W. LU, J.-J. ZHOU, J. SAKUMA. “**Non-interactive and Output Expressive Private Comparison from Homomorphic Encryption**”. In: *ACM Asia Conference on Computer and Communications Security (AsiaCCS’18)*. ACM, 2018, pp. 67–74.
- [MES18] A. MOGHIMI, T. EISENBARTH, B. SUNAR. “**MemJam: A False Dependency Attack Against Constant-Time Crypto Implementations in SGX**”. In: *The Cryptographers’ Track at the RSA Conference (CT-RSA’18)*. Vol. 10808. LNCS. Springer, 2018, pp. 21–44.
- [MIE17] A. MOGHIMI, G. IRAZOQUI, T. EISENBARTH. “**CacheZoom: How SGX Amplifies the Power of Cache Attacks**”. In: *Cryptographic Hardware and Embedded Systems (CHES’17)*. Vol. 10529. LNCS. Springer, 2017, pp. 69–90.

- [MLS<sup>+</sup>20] P. MISHRA, R. LEHMKUHL, A. SRINIVASAN, W. ZHENG, R. A. POPA. “**Delphi: A Cryptographic Inference Service for Neural Networks**”. In: *USENIX Security Symposium’20*. USENIX, 2020, pp. 2505–2522.
- [MNPS04] D. MALKHI, N. NISAN, B. PINKAS, Y. SELLA. “**Fairplay – A Secure Two-Party Computation System**”. In: *USENIX Security Symposium 2004*. USENIX Association, 2004, pp. 287–302.
- [MR18] P. MOHASSEL, P. RINDAL. “**ABY<sup>3</sup>: A Mixed Protocol Framework for Machine Learning**”. In: *ACM Conference on Computer and Communications Security (CCS’18)*. ACM, 2018, pp. 35–52.
- [MS13] P. MOHASSEL, S. S. SADEGHIAN. “**How to Hide Circuits in MPC – An Efficient Framework for Private Function Evaluation**”. In: *Advances in Cryptology – EUROCRYPT’13*. Vol. 7881. LNCS. Springer, 2013, pp. 557–574.
- [MSS14] P. MOHASSEL, S. S. SADEGHIAN, N. P. SMART. “**Actively Secure Private Function Evaluation**”. In: *Advances in Cryptology – ASIACRYPT’14*. Vol. 8874. LNCS. Springer, 2014, pp. 486–505.
- [MZ17] P. MOHASSEL, Y. ZHANG. “**SecureML: A System for Scalable Privacy-Preserving Machine Learning**”. In: *IEEE Symposium on Security and Privacy (S&P’17)*. IEEE, 2017, pp. 19–38.
- [NSMS14] S. NIKSEFAT, B. SADEGHIYAN, P. MOHASSEL, S. S. SADEGHIAN. “**ZIDS: A Privacy-Preserving Intrusion Detection System Using Secure Two-Party Computation Protocols**”. In: *The Computer Journal* 57.4 (2014). Oxford University Press, pp. 494–509.
- [Pin02] B. PINKAS. “**Cryptographic Techniques for Privacy-Preserving Data Mining**”. In: *SIGKDD Explorations* 4.2 (2002). ACM, pp. 12–19.
- [PKV<sup>+</sup>14] V. PAPPAS, F. KRELL, B. VO, V. KOLESNIKOV, T. MALKIN, S. G. CHOI, W. GEORGE, A. D. KEROMYTIS, S. BELLOVIN. “**Blind Seer: A Scalable Private DBMS**”. In: *IEEE Symposium on Security and Privacy (S&P’14)*. IEEE, 2014, pp. 359–374.
- [PSS09] A. PAUS, A.-R. SADEGHI, T. SCHNEIDER. “**Practical Secure Evaluation of Semi-Private Functions**”. In: *Applied Cryptography and Network Security (ACNS’09)*. Vol. 5536. LNCS. Springer, 2009, pp. 89–106.
- [PVS<sup>+</sup>19] R. PEHARZ, A. VERGARI, K. STELZNER, A. MOLINA, M. TRAPP, X. SHAO, K. KERSTING, Z. GHAHRAMANI. “**Random Sum-Product Networks: A Simple and Effective Approach to Probabilistic Deep Learning**”. In: *Conference on Uncertainty in Artificial Intelligence (UAI’19)*. AUAI Press, 2019, p. 124.
- [RMD18] A. RAGO, C. MARCOS, J. A. DIAZ-PACE. “**Using Semantic Roles to Improve Text Classification in the Requirements Domain**”. In: *Language Resources and Evaluation* 52.3 (2018). Springer, pp. 801–837.
- [RRK18] B. D. ROUHANI, M. S. RIAZI, F. KOUSHANFAR. “**Deepsecure: Scalable Provably-Secure Deep Learning**”. In: *Design Automation Conference (DAC’18)*. ACM, 2018, 2:1–2:6.
- [RSC<sup>+</sup>19] M. S. RIAZI, M. SAMRAGH, H. CHEN, K. LAINE, K. E. LAUTER, F. KOUSHANFAR. “**XONN: XNOR-based Oblivious Deep Neural Network Inference**”. In: *USENIX Security Symposium 2019*. USENIX Association, 2019, pp. 1501–1518.



- [RWT<sup>+</sup>18] M. S. RIAZI, C. WEINERT, O. TKACHENKO, E. M. SONGHORI, T. SCHNEIDER, F. KOUSHANFAR. “**Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications**”. In: *ACM Asia Conference on Computer and Communications Security (AsiaCCS’18)*. ACM, 2018, pp. 707–721.
- [Sch08] T. SCHNEIDER. “**Practical Secure Function Evaluation**”. MA thesis. University Erlangen-Nürnberg, Germany, 2008.
- [Sch11] T. SCHNEIDER. “**Engineering Secure Two-Party Computation Protocols**”. PhD thesis. Ruhr-Universität Bochum, Germany, 2011.
- [SLKP17] M.-W. SHIH, S. LEE, T. KIM, M. PEINADO. “**T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs**”. In: *Network and Distributed System Security Symposium (NDSS’17)*. The Internet Society, 2017.
- [SS08] A.-R. SADEGHI, T. SCHNEIDER. “**Generalized Universal Circuits for Secure Evaluation of Private Functions with Application to Data Classification**”. In: *Information Security and Cryptology (ICISC’08)*. Vol. 5461. LNCS. Springer, 2008, pp. 336–353.
- [SWG<sup>+</sup>17] M. SCHWARZ, S. WEISER, D. GRUSS, C. MAURICE, S. MANGARD. “**Malware Guard Extension: Using SGX to Conceal Cache Attacks**”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA’17)*. Vol. 10327. LNCS. Springer, 2017, pp. 3–24.
- [SY99] T. SANDER, A. L. YOUNG, M. YUNG. “**Non-Interactive CryptoComputing For NC<sup>1</sup>**”. In: *Foundations of Computer Science (FOCS’99)*. IEEE, 1999, pp. 554–567.
- [TASF09] A. K. TANWANI, M. J. AFRIDI, M. Z. SHAFIQ, M. FAROOQ. “**Guidelines to Select Machine Learning Scheme for Classification of Biomedical Datasets**”. In: *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO’09)*. Vol. 5483. LNCS. Springer, 2009, pp. 128–139.
- [TBK20] A. TUENO, Y. BOEV, F. KERSCHBAUM. “**Non-Interactive Private Decision Tree Evaluation**”. In: *Data and Applications Security and Privacy (DBSec’20)*. Vol. 12122. LNCS. Springer, 2020, pp. 174–194.
- [TKK19] A. TUENO, F. KERSCHBAUM, S. KATZENBEISSER. “**Private Evaluation of Decision Trees using Sublinear Cost**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2019.1* (2019). De Gruyter Open, pp. 266–286.
- [TMW<sup>+</sup>20] A. TREIBER, A. MOLINA, C. WEINERT, T. SCHNEIDER, K. KERSTING. “**CryptoSPN: Privacy-preserving Sum-Product Network Inference**”. In: *European Conference on Artificial Intelligence (ECAI’20)*. CEUR-WS.org, 2020.
- [TMZC17] R. K. H. TAI, J. P. K. MA, Y. ZHAO, S. S. M. CHOW. “**Privacy-Preserving Decision Trees Evaluation via Linear Functions**”. In: *European Symposium on Research in Computer Security (ESORICS’17)*. Vol. 10493. LNCS. Springer, 2017, pp. 494–512.
- [Val76] L. G. VALIANT. “**Universal Circuits (Preliminary Report)**”. In: *ACM Symposium on Theory of Computing (STOC’76)*. ACM, 1976, pp. 196–203.
- [WCP<sup>+</sup>17] W. WANG, G. CHEN, X. PAN, Y. ZHANG, X. WANG, V. BINDSCHAEDLER, H. TANG, C. A. GUNTER. “**Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX**”. In: *ACM Conference on Computer and Communications Security (CCS’17)*. ACM, 2017, pp. 2421–2434.
- [Weg87] I. WEGENER. “**The complexity of Boolean functions**”. Wiley-Teubner, 1987.



- [WFNL16] D. J. WU, T. FENG, M. NAEHRIG, K. E. LAUTER. **“Privately Evaluating Decision Trees and Random Forests”**. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2016.4 (2016). De Gruyter Open, pp. 335–355.
- [Woe05] P. WOELFEL. **“Bounds on the OBDD-size of integer multiplication via universal hashing”**. In: *Journal of Computer and System Sciences* 71.4 (2005), pp. 520–534.
- [XCP15] Y. XU, W. CUI, M. PEINADO. **“Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems”**. In: *IEEE Symposium on Security and Privacy (S&P’15)*. IEEE, 2015, pp. 640–656.
- [XLH<sup>+</sup>20] L. XUE, D. LIU, C. HUANG, X. LIN, X. S. SHEN. **“Secure and Privacy-Preserving Decision Tree Classification with Lower Complexity”**. In: *Journal of Communications and Information Networks* 5.1 (2020). IEEE, pp. 16–25.
- [Yao82] A. C.-C. YAO. **“Protocols for Secure Computations (Extended Abstract)”**. In: *Foundations of Computer Science (FOCS’82)*. IEEE, 1982, pp. 160–164.
- [Yao86] A. C.-C. YAO. **“How to Generate and Exchange Secrets (Extended Abstract)”**. In: *Foundations of Computer Science (FOCS’86)*. IEEE, 1986, pp. 162–167.
- [ZDW19] Y. ZHENG, H. DUAN, C. WANG. **“Towards Secure and Efficient Outsourcing of Machine Learning Classification”**. In: *European Symposium on Research in Computer Security (ESORICS’19)*. Vol. 11735. LNCS. Springer, 2019, pp. 22–40.
- [ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **“Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates”**. In: *Advances in Cryptology – EUROCRYPT’15*. Vol. 9057. LNCS. Springer, 2015, pp. 220–250.
- [ZYZL19] S. ZHAO, Y. YU, J. ZHANG, H. LIU. **“Valiant’s Universal Circuits Revisited: an Overall Improvement and a Lower Bound”**. In: *Advances in Cryptology – ASIACRYPT’19*. Vol. 11921. LNCS. Springer, 2019, pp. 401–425.

## List of Figures

---

1.1	Variants of secure function evaluation (SFE) and private function evaluation (PFE). . . . .	2
2.1	Timeline of research on universal circuits (UCs) and the corresponding UC sizes for private circuits with $n$ gates. . . . .	11
3.1	Decision tree with $m = 2$ decision nodes and $d = 2$ depth. . . . .	15

## List of Tables

---

2.1	Comparison of the concrete communication and computation of passively secure private function evaluation approaches with $\mathcal{O}(n \log n)$ asymptotic complexity. For large enough values of the private circuit size $n$ , the homomorphic encryption (HE)-based private function evaluation (PFE) protocols [KM11; MS13; BBKL20] with linear complexity $\mathcal{O}(n)$ (cf. Section 2.2.2) have the lowest computation and communication [HKRS20]. . . . .	13
3.1	Constant-round protocols composed of the presented sub-protocols for selection, comparison and path evaluation based on homomorphic encryption (H) and garbling techniques (G). We depict if a protocol requires dummy decision nodes to be introduced in order to pad each path to the maximum depth of the decision tree and we show the number of rounds of each protocol. . . .	17
3.2	Summary of private decision tree evaluation protocols in the client-server setting. Table modified and extended from [TKK19; TBK20], assuming that the number of input features is $\mathcal{O}(m)$ in the number of decision nodes $m$ . $d$ denotes the depth of the decision tree. For the underlying tools, we use the following notation: fully homomorphic encryption (FHE), homomorphic encryption (HE), garbled circuit (GC), oblivious transfer (OT), secret sharing (SS), function secret sharing (FSS), oblivious random access memory (ORAM), oblivious data structure (ODS). . . . .	20

## List of Abbreviations

---

<b>BDD</b>	binary decision diagram
<b>DBMS</b>	database management system
<b>FHE</b>	fully homomorphic encryption
<b>FSS</b>	function secret sharing
<b>GC</b>	garbled circuit
<b>HE</b>	homomorphic encryption
<b>ODS</b>	oblivious data structure
<b>ORAM</b>	oblivious random access memory
<b>OT</b>	oblivious transfer
<b>PFE</b>	private function evaluation
<b>semi-PFE</b>	semi-private function evaluation
<b>SFE</b>	secure function evaluation
<b>SMPC</b>	secure multi-party computation
<b>SPN</b>	sum-product network
<b>SS</b>	secret sharing
<b>TEE</b>	trusted execution environment
<b>UC</b>	universal circuit

## List of Own Publications

---

### Peer-reviewed Publications

- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Springer. Online version: <https://ia.cr/2019/348>. Code: <https://crypto.de/code/UC>, pp. 1216–1271. CORE Rank A\*. Appendix C.
- [BDF<sup>+</sup>20] J. BUCHMANN, G. DESSOUKY, T. FRASSETTO, Á. KISS, A.-R. SADEGHI, T. SCHNEIDER, G. TRAVERSO, S. ZEITOUNI. “**SAFE: A Secure and Efficient Long-Term Distributed Storage System**”. In: 8. *ACM International Workshop on Security in Blockchain and Cloud Computing (SBC’20)*. Full version: <https://ia.cr/2020/690>. To appear. ACM, 2020.
- [FKSW19] S. FELSEN, Á. KISS, T. SCHNEIDER, C. WEINERT. “**Secure and Private Function Evaluation with Intel SGX**”. In: 10. *ACM Cloud Computing Security Workshop (CCSW’19)*. Online version: <https://ia.cr/20>. ACM, 2019, pp. 165–181.
- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: 23. *Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Full version: <https://ia.cr/2017/798>. Code: <https://crypto.de/code/UC>. Springer, 2017, pp. 443–470. CORE Rank A. Appendix B.
- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: 25. *European Symposium on Research in Computer Security (ESORICS’20)*. Vol. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://crypto.de/code/linearPFE>. Springer, 2020, pp. 401–420. CORE Rank A. Appendix D.
- [JKS<sup>+</sup>18] K. JÄRVINEN, Á. KISS, T. SCHNEIDER, O. TKACHENKO, Z. YANG. “**Faster Privacy-Preserving Location Proximity Schemes**”. In: 17. *International Conference on Cryptology And Network Security (CANS’18)*. Vol. 11124. LNCS. Full version: <https://ia.cr/2018/694>. Springer, 2018, pp. 3–22. CORE Rank B.
- [JKS<sup>+</sup>19] K. JÄRVINEN, Á. KISS, T. SCHNEIDER, O. TKACHENKO, Z. YANG. “**Faster Privacy-Preserving Location Proximity Schemes for Circles and Polygons**”. In: *IET Information Security* 14.3 (2019). IET, pp. 254–265. CORE Rank C.

- [KKRS16] Á. KISS, J. KRÄMER, P. RAUZY, J.-P. SEIFERT. “**Algorithmic Countermeasures Against Fault Attacks and Power Analysis for RSA-CRT**”. In: 7. *International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE’16)*. Vol. 9689. LNCS. Springer, 2016, pp. 111–129.
- [KKS15] Á. KISS, J. KRÄMER, A. STÜBER. “**On the Optimality of Differential Fault Analyses on CLEFIA**”. In: 6. *International Conference on Mathematical Aspects of Computer and Information Sciences (MACIS’15)*. Vol. 9582. LNCS. Springer, 2015, pp. 181–196. CORE Rank C.
- [KLS<sup>+</sup>17] Á. KISS, J. LIU, T. SCHNEIDER, N. ASOKAN, B. PINKAS. “**Private Set Intersection for Unequal Set Sizes with Mobile Applications**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2017.4* (2017). De Gruyter Open. Full version: <https://ia.cr/2017/670>. Code: <https://crypto.de/code/MobilePSI>, pp. 177–197. CORE Rank B.
- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs) 2019.2* (2019). De Gruyter Open. Online version: <https://ia.cr/2018/1099>. Code: <https://crypto.de/code/PDTE>, pp. 187–208. CORE Rank B. Appendix E.
- [KSS19] Á. KISS, O. SCHICK, T. SCHNEIDER. “**Web Application for Privacy-Preserving Scheduling using Secure Computation**”. In: 16. *International Conference on Security and Cryptography (SECRYPT’19)*. Short paper. Code: <https://crypto.de/code/scheduling>. SciTePress, 2019, pp. 456–463. CORE Rank B.
- [KS16] Á. KISS, T. SCHNEIDER. “**Valiant’s Universal Circuit is Practical**”. In: 35. *Advances in Cryptology – EUROCRYPT’16*. Vol. 9665. LNCS. Full version: <https://ia.cr/2016/093>. Code: <https://crypto.de/code/UC>. Springer, 2016, pp. 699–728. CORE Rank A\*. Appendix A.

## CURRICULUM VITAE

### PERSONAL DATA

---

Name                      Ágnes KISS  
Phone:                    +49 1590 8114675  
Email:                    [kiss@encrypto.cs.tu-darmstadt.de](mailto:kiss@encrypto.cs.tu-darmstadt.de)

### WORK EXPERIENCE AND EDUCATION

---

FEB 2015 – PRESENT    **Research Assistant and Doctoral Student**  
I am a research assistant in the **Cryptography and Privacy Engineering Group (ENCRYPTO)** headed by Prof. Dr. Thomas Schneider, pursuing my doctoral study at the **Technical University of Darmstadt** (Germany). My main responsibilities include research activities with the goal of publishing at relevant top conferences, helping in the management of projects that our group is part of and supervising undergraduate theses.  
Maternity leaves: 2019/09-12, 2020/01-03, 2020/07-09.

#### ADDITIONAL ACTIVITIES

JAN 2015 – PRESENT    I became an active member of the alumni community of the EIT Digital Master School. I served as the treasurer of the **EIT Digital Alumni Foundation** between January 2015 and July 2016.

SEP 2012 – NOV 2014    **Master Degree**  
I completed a double degree program in SECURITY AND PRIVACY (Computer Science) in the **EIT ICT Labs (now EIT Digital) Master School** with the first year at the **University of Trento** (Italy) and the second year at the **Technical University of Berlin** (Germany) with a minor in INNOVATION AND ENTREPRENEURSHIP.

MAY 2014 – AUG 2014    **Internship and Master thesis at Deutsche Telekom**  
I was an intern and a Master thesis student at the **Security in Telecommunications Department** in Berlin supervised by Dr. Juliane Krämer and Prof. Dr. Jean-Pierre Seifert. I completed my Master thesis with the title “*Testing Self-Secure Exponentiation Countermeasures against the Bellcore Attack*”, which received good evaluation (CAST-Förderpreis IT-Sicherheit 2015 3rd place, publication at COSADE’16).

SEP 2009 – JUN 2012    **Bachelor Degree**  
I finished my Bachelor studies in MATHEMATICS, with a specialization in Applied Mathematics, at the **Budapest University of Technology and Economics** (Hungary).

SEP 2011 – JUN 2012    With the supervision of Dr. Tamás Fleiner, I studied minimum cuts of graphs for my bachelor thesis titled *Finding and Representations of Minimum Cuts and Their Applications*, which received good evaluation.

## AWARDS, STIPENDS

---

- DEC 2019 Together with Ghada Dessouky, Tommaso Frassetto, Giulia Traverso, Shaza Zeitouni, we have recieved the **CROSSING Collaboration Award** for our paper titled *SAFE: A Secure and Efficient Long-Term Distributed Storage System* for our internal collaborative work within the project CROSSING.
- SEP 2018 I was selected to be one of the 200 international young researchers at the **6. Heidelberg Laureate Forum (HLF)**, which brings together graduate and undergraduate students with recipients of the most prestigious awards in mathematics and computer science, i.e., the Abel Prize, ACM A.M. Turing Award, ACM Prize in Computing, Fields Medal and the Nevanlinna Prize.
- JUN 2018 I received a stipend for visiting the **Women in Theory (WIT) Workshop**, intended for graduate and exceptional undergraduate students in the area of theory of computer science.
- MAY 2015 I was a finalist for the **CAST MSc Award** in IT Security (CAST Förderpreis IT-Sicherheit), and obtained the third price with my Master thesis titled *Testing Self-Secure Exponentiation Countermeasures Against the Bellcore Attack*.

## PROFESSIONAL SERVICE

---

- JUN 2019 – NOV 2019 I served as a **Program Committee member** in the PPML'19 international workshop.
- MAY 2015 – PRESENT I served as an **external reviewer** in the following international conferences: NDSS'21, CCS'19, ASIACRYPT'19, IH&MMSec'19, CANS'18, AsiaCCS'18, EUROCRYPT'18, INDOCRYPT'17, CANS'17, ASIACRYPT'17, ACNS'17, ASIACCS'17, IH&MMSec'16, ACNS'16, IEEE S&P'16, EUROCRYPT'16.
- I served as a **reviewer** in the following international journals: TIFS 2019, FGCS 2018.

# Appendices



## A Valiant's Universal Circuit is Practical (EUROCRYPT'16)

---

- [KS16] Á. KISS, T. SCHNEIDER. “Valiant's Universal Circuit is Practical”. In: 35. *Advances in Cryptology – EUROCRYPT'16*. Vol. 9665. LNCS. Full version: <https://ia.cr/2016/093>. Code: <https://encrypto.de/code/UC>. Springer, 2016, pp. 699–728. CORE Rank A\*. Appendix A.

[https://doi.org/10.1007/978-3-662-49890-3\\_27](https://doi.org/10.1007/978-3-662-49890-3_27)

# Valiant’s Universal Circuit is Practical

Ágnes Kiss<sup>(✉)</sup> and Thomas Schneider

TU Darmstadt, Darmstadt, Germany  
{agnes.kiss,thomas.schneider}@crisp-da.de

**Abstract.** Universal circuits (UCs) can be programmed to evaluate any circuit of a given size  $k$ . They provide elegant solutions in various application scenarios, e.g. for private function evaluation (PFE) and for improving the flexibility of attribute-based encryption (ABE) schemes. The optimal size of a universal circuit is proven to be  $\Omega(k \log k)$ . Valiant (STOC’76) proposed a size-optimized UC construction, which has not been put in practice ever since. The only implementation of universal circuits was provided by Kolesnikov and Schneider (FC’08), with size  $\mathcal{O}(k \log^2 k)$ .

In this paper, we refine the size of Valiant’s UC and further improve the construction by (at least)  $2k$ . We show that due to recent optimizations and our improvements, it is the best solution to apply in the case for circuits with a constant number of inputs and outputs. When the number of inputs or outputs is linear in the number of gates, we propose a more efficient hybrid solution based on the two existing constructions. We validate the practicality of Valiant’s UC, by giving an example implementation for PFE using these size-optimized UCs.

**Keywords:** Universal circuit · Size-optimization · Private function evaluation

## 1 Introduction

Any computable function  $f(x)$  can be represented as a Boolean circuit with input bits  $x = (x_1, \dots, x_u)$ . Universal circuits (UCs) are programmable circuits, which means that beyond the true  $u$  inputs, they receive  $p = (p_1, \dots, p_m)$  program bits as further inputs. By means of these program bits, the universal circuit is programmed to evaluate the function, such that  $UC(x, p) = f(x)$ . The advantage of universal circuits in general is that one can apply the same UC for computing different functions of the same size. An analogy between universal circuits and a universal Turing machine allows to turn any function into data in the form of a program description. Thus, the size-depth problem of UCs can be related to the time-space problem for Turing machines [Val76].

Efficient constructions considering both the size and the depth of the UC were proposed. The first approach was the optimization of the size by Valiant [Val76], resulting in a construction with asymptotically optimal size  $\mathcal{O}(k \log k)$  and depth  $\mathcal{O}(k)$ , where  $k$  denotes the size of the simulated circuits. The second optimization

was proposed with respect to the UC depth in [CH85], where a construction with linear depth  $\mathcal{O}(d)$  in the simulated circuit depth  $d$  and size  $\mathcal{O}(\frac{k^3 d}{\log k})$  was designed. In this paper, due to the applications that we revisit in Sect. 1.2, e.g., diagnostic programs, blinded policies and database queries, we concentrate on the existing size-optimized UCs and note, that the asymptotically optimal size is  $\Omega(k \log k)$  [Val76, Weg87].

The most prominent application of universal circuits is the evaluation of private functions based on *secure function evaluation* (SFE) or *secure two-party computation*. SFE enables two parties  $P_1$  and  $P_2$  to evaluate a publicly known function  $f(x, y)$  on their private inputs  $x$  and  $y$ , ensuring that none of the participants learns anything about the other participant's input. SFE ensures that both  $P_1$  and  $P_2$  learn the correct result of the evaluation. Many secure computation protocols use Boolean circuits for representing the desired functionality, such as Yao's garbled circuit protocol [Yao86, LP09a] and the GMW protocol [GMW87]. In some applications the function itself should be kept secret. This setting is called *private function evaluation* (PFE), where we assume that only one of the parties  $P_1$  knows the function  $f(x)$ , whereas the other party  $P_2$  provides the input to the private function.  $P_2$  learns no information about  $f$  besides the size of the circuit defining the function and the number of inputs and outputs.

PFE can be reduced to SFE [AF90, SYY99, Pin02, KS08b] by securely evaluating a UC that is programmed by  $P_1$  to evaluate the function  $f$  on  $P_2$ 's input  $x$ . Thus,  $P_1$  provides the program bits for the UC and  $P_2$  provides his private input  $x$  into an SFE protocol that computes a UC. The complexity of PFE in this case is determined mainly by the complexity of the UC construction. The security follows from that of the SFE protocol that is used to evaluate the UC. If the SFE protocol is secure against semi-honest, covert or malicious adversaries, then the PFE protocol is secure in the same adversarial setting.

### 1.1 Related Work on Universal Circuits and Private Function Evaluation

*Universal Circuits.* Valiant presented an asymptotically optimal universal circuit construction with size  $\approx 4.75(u + v + k^*) \log_2(u + v + k^*)$  [Val76], relying on edge-universal graphs.  $u$ ,  $k$  and  $v$  denote the respective number of inputs, gates and outputs in the simulated circuit, and  $k^*$  is the number of gates in the equivalent fanout-2 circuit, with  $k \leq k^* \leq 2k + v$ . Valiant's size-optimized UC construction was recapitulated in [Weg87, Sect. 4.8]. However, Valiant's construction has been considered to be mostly a proof of existence of a universal circuit, whereas details needed for the practical realization, e.g., how to derive the program for the UC are left open. Kolesnikov and Schneider proposed a UC construction with size  $\approx 0.75k \log_2^2 k + 2.25k \log_2 k + k \log_2 u + (0.5k + 0.5v) \log_2 v$  [KS08b, Sch08]. They present the first implementation of PFE using UCs by extending the Fairplay secure computation framework [MNPS04]. Some building blocks of this construction are of interest, but due to its asymptotically non-optimal size, we show in Sect. 3.2 that Valiant's UC construction results in smaller UCs for circuits in

the most general case. The UC constructions from [Val76,KS08b] were generalized for circuits consisting of gates with more than two inputs in [SS08]. In this paper, we show the practicality of Valiant’s UC construction.

In concurrent and independent work [LMS16], Lipmaa et al. also bring the same UC construction to practice. They detail a  $k$ -way recursive construction for UCs, instantiate it for  $k \in \{2, 4\}$  as in [Val76], and decrease its *total* number of gates compared to that of Valiant’s construction. However, in contrast to our optimizations, their number of AND gates is exactly the same and therefore their improvement does not affect PFE with UC, when XOR gates are evaluated for free [KS08a]. Currently their implementation for generating and programming UCs supports the 2-way recursive construction, the same construction that we study and realize in practice in this work.

*Private Function Evaluation.* In [KM11], Katz and Malka presented an approach for PFE that does not rely on UCs. They use (singly) homomorphic public-key encryption as well as a symmetric-key encryption scheme and achieve constant-round PFE with linear communication complexity. However, the number of public-key operations is linear in the circuit size and due to the gap between the efficiency of public-key and symmetric-key operations, this results in a less efficient protocol for circuits with reasonable size. Their protocol is secure against semi-honest adversaries and uses Yao’s garbled circuit technique [Yao86]. Mohassel and Sadeghian consider PFE with semi-honest adversaries in [MS13]. Their generic PFE framework can be instantiated with different secure computation protocols. The first version uses homomorphic encryption with which they achieve linear complexity in the circuit size and the second alternative relies solely on oblivious transfers (OT), that results in a method with  $\mathcal{O}(k \log k)$  symmetric-key operations, where  $k$  denotes the circuit size. The OT-based construction is more desirable in practice, since using OT extension, the number of expensive public-key operations can significantly be reduced, s.t. it is independent of the number of OTs [IKNP03, ALSZ13]. The asymptotical complexity of the OT-based construction of [MS13] and Valiant’s UCs for PFE is the same, and therefore we compare these solutions for PFE in more detail in Sect. 4.2. Mohassel et al. extend the framework from [MS13] to malicious adversaries in [MSS14] and show that an actively secure PFE framework with linear complexity  $\mathcal{O}(k)$  is feasible, using singly homomorphic encryption.

## 1.2 Applications of Universal Circuits

Universal circuits have several applications, which we summarize in this section.

*Private Function Evaluation.* As mentioned before, UCs can be used to securely evaluate a private function using a generic secure computation protocol. [CCKM00] shows an application for secure computation, where evaluating UCs or other PFE protocols would ensure privacy: when *autonomous mobile agents* migrate between several distrusting hosts, the privacy of the inputs of the hosts is achieved using SFE, while privacy of the mobile agent’s code can be

guaranteed with PFE. Privacy-preserving *credit checking* using garbled circuits is described in [FAZ05]. Their original scheme cannot represent any policy, though by evaluating a UC, their scheme can be extended to more complicated credit checking policies. [OI05] show a method to *filter remote streaming data* obliviously, using secret keywords and their combinations. Their scheme can additionally preserve data privacy by using PFE to search the matching data with a private search function. Privacy-preserving evaluation of *diagnostic programs* was considered in [BPSW07], where the owner of the program does not want to reveal the diagnostic method and the user does not want to reveal his data. Example applications for such programs include medical systems [BFK+09] and remote software fault diagnosis, where in both cases the function and the user's input are desired to be handled privately. In the protocol presented in [BPSW07], the diagnostic programs are represented as binary decision trees or branching programs which can easily be converted into a Boolean circuit representation and evaluated using PFE based on universal circuits. Besides, PFE can be applied to create *blinded policy evaluation protocols* [FAL06, FLA06]. [FAL06] utilizes UCs for so-called oblivious circuit policies and [DDKZ13] for hiding the circuit topology in order to create one-time programs. Since PFE using UCs utilizes general secure computation protocols, it is possible to outsource the function and the data to two or multiple servers (using XOR secret sharing) and then run private queries on these. This is not directly possible with other PFE protocols, e.g., with the protocol presented in [KM11] or the homomorphic encryption-based protocols from [MS13, MSS14].

*Beyond Private Function Evaluation.* Besides being used for PFE, UCs can be applied in various other scenarios. Efficient *verifiable computation* on encrypted data was studied in [FGP14]. A verifiable computation scheme was proposed for arbitrary computations and a UC is required to hide the function. [GGPR13] make use of universal circuits for reducing the verifier's preprocessing step. In [GHV10], a *multi-hop homomorphic encryption* scheme is proposed that also uses a universal circuit evaluator to achieve the privacy of the function. When the common reference string is dependent on a function that the verifier is interested in outsourcing, then the function description can be provided as input to a UC of appropriate size. In [PKV+14, FVK+15], universal circuits are used for hiding *queries in database management systems* (DBMSs). The Blind Seer DBMS was improved in [PKV+14] by making use of a simpler UC for evaluating queries, which does not hide the circuit topology. The authors mention that in case the topology of the SQL formula and the circuit have to be kept private, a UC can be utilized. As described in [Att14], the *Attribute-Based Encryption* (ABE) schemes for some polynomial-size circuits can be turned into ciphertext-policy ABE by using universal circuits. The ABE scheme of [GGHZ14] also uses UCs.

### 1.3 Outline and Our Contributions

In Sect. 2, we revisit the two existing size-optimized UC constructions of [Val76, KS08b]. We put an emphasis on the asymptotically size-optimal

method proposed by Valiant in [Val76]. This complex construction makes use of an internal graph representation and programs a so-called edge-universal graph. However, the algorithm for programming a universal circuit is not explicitly described and in the presence of the included optimizations is not straightforwardly applicable. In Sect. 2.1, we recapitulate Valiant's recursive edge-universal graph construction and describe how the construction of UCs can be reduced to this problem. In Sect. 2.2, we briefly summarize the main building blocks of the UC construction of [KS08b].

*Optimized Size and Depth of Valiant's UC Construction:* In Sect. 3, we elaborate on the concrete size of Valiant's UC construction. We refine upper and lower bounds for the size of the edge-universal graph and approximate a closed formula with  $\leq 2\%$  deviation from the actual size in Sect. 3.1. We include two optimizations detailed in Sect. 3.2, achieving altogether a linear improvement of at least  $4u+4v+2k$ . We give hybrid constructions for cases with many inputs and outputs in the same section. In Sect. 3.2, we compare the refined concrete size and the depth of Valiant's construction with that of [KS08b] and conclude the advantage of Valiant's method (potentially using building blocks from [KS08b]).

*Valiant's Size-Optimized UC Construction in Practice:* In Sect. 4, we detail the steps of our algorithm for a practical realization of Valiant's UC construction and provide an example application for PFE. We describe the internal representations and the algorithms in our UC compiler in Sect. 4.1, along with detailed implementations of universal gates and switches. We compare our resulting PFE with the OT-based protocol from [MS13] in Sect. 4.2. We show concrete example circuits and elaborate on the number of symmetric-key operations and the performance of our UC compiler.

## 2 Existing Universal Circuit Constructions

In this section, we summarize the two size-optimized universal circuit constructions: of [Val76] in Sect. 2.1 and of [KS08b] in Sect. 2.2.

### 2.1 Valiant's Universal Circuit Construction

In this section, we describe Valiant's edge-universal graph construction for graphs for which all nodes have at most one incoming and at most one outgoing edge and detail how two such graphs can be used for constructing universal circuits [Val76].

**Edge-Universal Graphs.**  $G = (V, E)$  is a directed graph with the set of nodes  $V = \{1, \dots, n\}$  and the set of edges  $E \subseteq V \times V$ . A directed graph has *fanin* or *fanout*  $\ell$  if each of its nodes has at most  $\ell$  edges directed into or out of it, respectively.  $\Gamma_\ell(n)$  denotes the set of all acyclic directed graphs with  $n$  nodes and fanin and fanout  $\ell$ . Further on, we require a labelling of the nodes in a



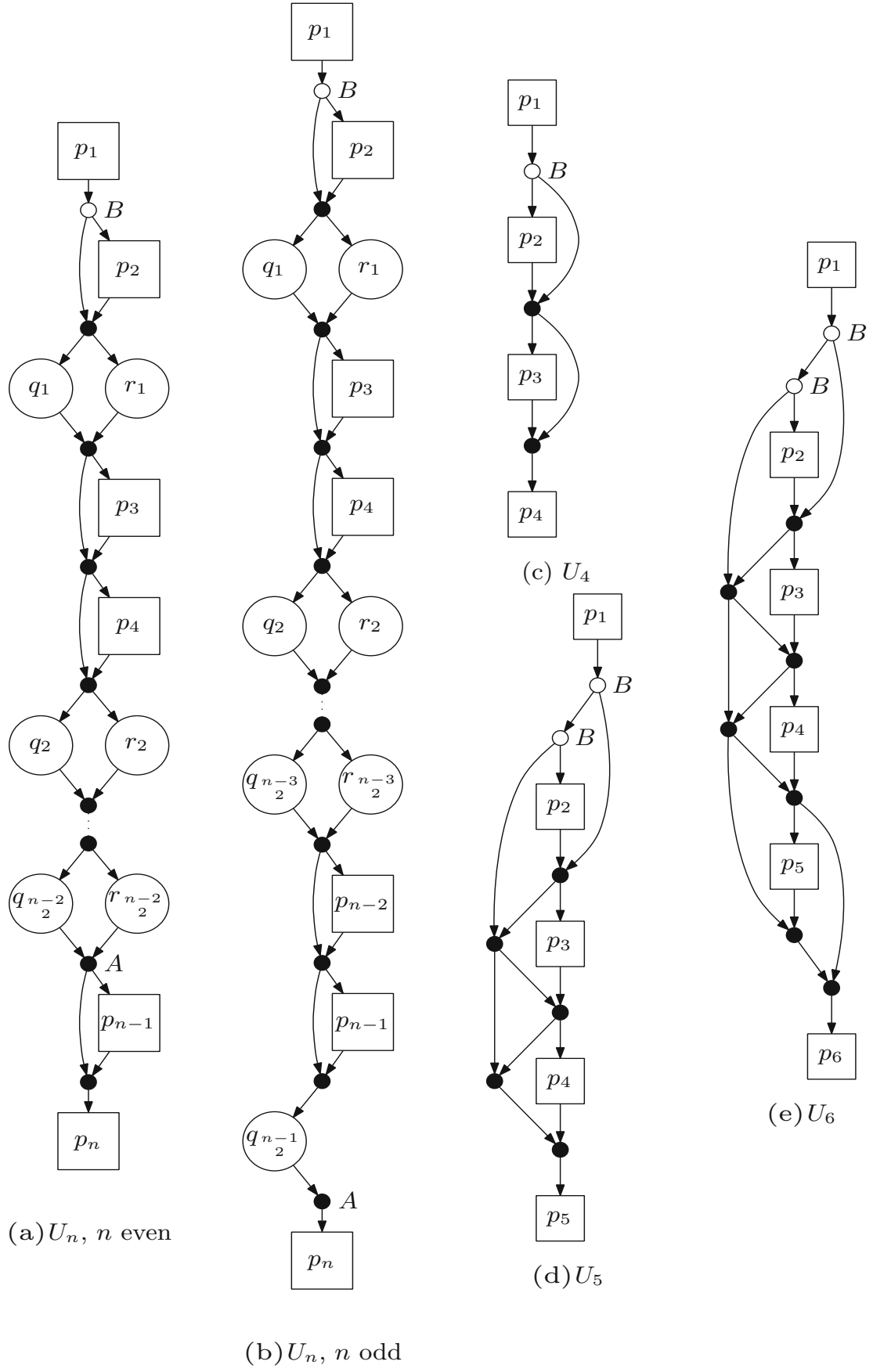
topological order, i.e.,  $i > j$  implies that there is no directed path from  $i$  to  $j$ . In a graph in  $\Gamma_\ell(n)$ , a topological ordering can always be found with computational complexity  $\mathcal{O}(n + \ell n)$ .

An *edge-embedding* of graph  $G = (V, E)$  into  $G' = (V', E')$  is a mapping that maps  $V$  into  $V'$  one-to-one, with possible additional nodes in  $V'$ , and  $E$  into directed paths in  $E'$ , such that they are pairwise edge-disjoint, i.e., an edge can be used only in one path. A graph  $G'$  is *edge-universal* for  $\Gamma_\ell(n)$  if it has distinguished poles  $\{p_1, \dots, p_n\} \subseteq V'$  and every graph  $G \in \Gamma_\ell(n)$  with node set  $V = \{1, \dots, n\}$  can be edge-embedded into  $G'$  by a mapping  $\varphi^G$  such that  $\varphi^G : i \mapsto p_i$  and  $\varphi^G : (i, j) \mapsto \{\text{path from pole } p_i \text{ to pole } p_j\}$  for each  $i, j \in V$ .

Here, we recapitulate Valiant's construction for acyclic edge-universal graph for  $\Gamma_1(n)$ , denoted by  $U_n$ , that has fewer than  $2.5n \log_2 n$  nodes, fanin and fanout 2 and poles with fanin and fanout 1. Valiant presents another edge-universal graph construction with a lower multiplicative constant  $2.375n \log_2 n$ . We omit that version of the algorithm for two reasons: firstly, our aim is to show the practicality of Valiant's approach and secondly, including all the optimizations even in the simpler construction is a challenging task in practice. The more efficient algorithm uses four subgraphs instead of two at each recursion and utilizes a skeleton with a more complex structure. For more details on this improved algorithm, the reader is referred to [Val76, LMS16]. We leave showing the practicality of the improved method as future work.

*Valiant's Edge-Universal Graph Construction for  $\Gamma_1(n)$  Graphs:* The edge-universal graph for  $\Gamma_1(n)$ , denoted by  $U_n$ , is constructed with poles  $\{p_1, \dots, p_n\}$  with fanin and fanout 1, which are connected according to the skeleton shown in Figs. 1a and b. The poles are emphasized as special nodes with squares, and the additional nodes are shown as circles. The recursive construction works as follows: the nodes denoted by  $\{q_1, \dots, q_{\lceil \frac{n-2}{2} \rceil}\}$  and  $\{r_1, \dots, r_{\lfloor \frac{n-2}{2} \rfloor}\}$  are considered as the poles of two smaller edge-universal graphs called subgraphs  $Q_{\lceil \frac{n-2}{2} \rceil}$  and  $R_{\lfloor \frac{n-2}{2} \rfloor}$ , respectively, that are otherwise not shown. Since they are poles of the two subgraphs with such a skeleton but not of  $U_n$ , they will have at most the allowed fanin and fanout 2: they inherit one incoming and one outgoing edge from the outer skeleton, and at most one incoming and one outgoing edge from the subgraph.  $Q_{\lceil \frac{n-2}{2} \rceil}$  (and  $R_{\lfloor \frac{n-2}{2} \rfloor}$ ) is then constructed similarly: the skeleton is completed and two smaller graphs with sizes  $\lceil \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rceil$  and  $\lfloor \frac{\lceil \frac{n-2}{2} \rceil - 2}{2} \rfloor$  (and sizes  $\lceil \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rceil$  and  $\lfloor \frac{\lfloor \frac{n-2}{2} \rfloor - 2}{2} \rfloor$ ) are constructed. For starting off the recursion,  $U_1$  is a graph with a single pole while  $U_2$  and  $U_3$  are graphs with two and three connected poles, respectively. Valiant gives special constructions for  $U_4, U_5$  and  $U_6$  and shows that it is possible to obtain the respective edge-universal graphs with altogether 3, 7 and 9 additional nodes, respectively, as shown in Figs. 1c, d, and e.

We recapitulate the proof from [Val76] that  $U_n$  is edge-universal for  $\Gamma_1(n)$ , such that any graph with  $n$  nodes and fanin and fanout 1 can be edge-embedded into  $U_n$ . According to the definition of edge-embedding, it has to be shown that given any  $\Gamma_1(n)$  graph  $G$  with set of edges  $E$ , for any  $(i, j) \in E$  and  $(k, l) \in E$



**Fig. 1.** Skeleton of Valiant's edge-universal graph and optimized cases.



we can find pairwise edge-disjoint paths from  $p_i$  to  $p_j$  and from  $p_k$  to  $p_l$  in  $U_n$ . As before, the labelling of nodes  $V = \{1, \dots, n\}$  in the  $\Gamma_1(n)$  graph is according to a topological order of the nodes.

Firstly, each two neighbouring poles of the edge-universal graph,  $p_{2s}$  and  $p_{2s+1}$  for  $s \in \{1, \dots, \lceil \frac{n}{2} \rceil\}$ , are thought of as merged superpoles, with their fanin and fanout becoming 2. In a similar manner, any  $G \in \Gamma_1(n)$  graph can be regarded as a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph with supernodes, i.e. each pair  $(2s, 2s+1)$  will be merged into one node in a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph  $G' = (V', E')$ . If there are edges between the nodes in  $G$ , they are simulated with loops<sup>1</sup>. The set of edges of this graph  $G$  is partitioned to sets  $E_1$  and  $E_2$ , s.t.  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$  are instances of  $\Gamma_1(\lceil \frac{n}{2} \rceil)$  and  $\Gamma_1(\lfloor \frac{n}{2} \rfloor)$ , respectively. This can be done efficiently, as shown later in this section. The edges in  $E_1$  are embedded as directed paths in  $Q$ , and the edges in  $E_2$  as directed paths in  $R$ . Both  $E_1$  and  $E_2$  have at most one edge directed into and at most one directed out of any supernode and therefore, there is only one edge from  $E_1$  and one from  $E_2$  to be simulated going through any superpole in  $U_n$  as well. Thus, the edge coming into a superpole  $(p_{2s}, p_{2s+1})$  in  $E_1$  is embedded as a path through  $q_{s-1}$ , while the edge going out of the pole in  $E_1$  is embedded as a path through  $q_s$  in the appropriate subgraph. Similarly, the edges in  $E_2$  are simulated as edges through  $r_{s-1}$  and  $r_s$ . These paths can be chosen disjoint according to the induction hypothesis. Finally, the paths from  $q_{s-1}$  and  $r_{s-1}$  to superpole  $(p_{2s-1}, p_{2s})$  as well as the paths from  $(p_{2s-1}, p_{2s})$  to  $q_s$  and  $r_s$  can be chosen edge-disjoint due to the skeleton shown in Figs. 1a and b. With this, Valiant's graph construction is a valid edge-universal graph construction with asymptotically optimal size  $\mathcal{O}(n \log n)$ , and depth  $\mathcal{O}(n)$  [Val76].

*Valiant's Edge-Universal Graph Construction for  $\Gamma_2(n)$  Graphs:* Given a directed acyclic graph  $G \in \Gamma_2(n)$ , the set of edges  $E$  can be separated into two distinct sets  $E_1$  and  $E_2$ , such that graphs  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$  are instances of  $\Gamma_1(n)$ , having fanin and fanout 1 for each node [Val76]. Given the set of nodes  $V = \{1, \dots, n\}$ , one constructs a bipartite graph  $\overline{G} = (\overline{V}, \overline{E})$  with nodes  $\overline{V} = \{m_1, \dots, m_n, m'_1, \dots, m'_n\}$  and edges  $\overline{E}$  such that  $(m_i, m'_j) \in \overline{E}$  if and only if  $(i, j) \in E$ . The edges of  $\overline{G}$  and thus the corresponding edges of  $G$  can be colored in a way that the result is a valid two-coloring. Having fanin and fanout at most 2, such coloring can be found directly with the following method, used in the proof of König-Hall theorem in [LP09b]:

- 1: **while** There are uncolored edges in  $\overline{G}$  **do**
- 2:     Choose an uncolored edge  $e = (m_i, m'_j)$  randomly and color the path or cycle that contains it in an alternating manner: the neighbouring edge(s) of an edge of the first color will be colored with the second color and vice versa.
- 3: **end while**

<sup>1</sup> We note that these  $G'$  graphs are constructed from the original  $\Gamma_1(n)$  graph  $G$  in order to define the correct embedding. Therefore, they are not required to be acyclic.

This coloring can be performed in  $\mathcal{O}(n)$  steps and it defines the edges in  $E_1$  and  $E_2$ , s.t.  $E_1$  contains the edges colored with color one and  $E_2$  the ones with color two and  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$  (cf. full version [KS16]).

With this method, the problem of constructing edge-universal graphs for  $\Gamma_2(n)$  can be reduced to the  $\Gamma_1(n)$  construction. After constructing two edge-universal graphs for  $\Gamma_1(n)$  (i.e.  $U_{n,1}$  and  $U_{n,2}$ ), their poles are merged and an edge-universal graph for  $\Gamma_2(n)$  is obtained. The merged poles now have fanin and fanout 2, since the poles of  $U_{n,1}$  and  $U_{n,2}$  previously had fanin and fanout 1.  $E_1$  can then be edge-embedded using the edges of  $U_{n,1}$  and  $E_2$  using the edges of  $U_{n,2}$ .

**Universal Circuits.** We now describe how to construct UCs by means of Valiant's edge-universal graph construction for  $\Gamma_2(n)$  graphs [Val76]. Our goal is to obtain an acyclic circuit built from special gates that simulate any acyclic Boolean circuit with  $u$  inputs,  $v$  outputs and  $k$  gates. In the circuit, the inputs of the gates are either connected to an input variable, to the output of another gate or are assigned a fixed constant. Due to the nature of Valiant's edge-universal graph construction, we have two restrictions on the original circuit. Firstly, all the gates must have at most two inputs and secondly, the fanout of inputs and gates must be at most 2, i.e., each input of the circuit and each output of any gate can only be the input of at most two later gates. This is necessary in order to guarantee that the graph of the original circuit has fanin and fanout 2. We note that the first restriction was present in case of the construction in [KS08b] as well, but the output of any input or any gate could be used multiple times. However, it was proven in [Val76] that the general case, where the fanout of the circuit can be any integer  $m \geq 2$ , can be transformed to the special case when  $m \leq 2$  by introducing copy gates, where the resulting circuit will have  $k^*$  gates with  $k \leq k^* \leq 2k + v$ , where  $k$  denotes the number of gates and  $v$  the number of outputs in the circuit. We detail how this can be done in Sect. 4.1.

After this transformation, given a circuit  $C$  with  $u$  inputs,  $v$  outputs and  $k^*$  gates with fanin and fanout 2, the graph of  $C$ , denoted by  $G^C$  consists of a node for each gate, input and output variable and thus is in  $\Gamma_2(u + v + k^*)$ . The wires of circuit  $C$  are represented by edges in  $G^C$ . A *topological ordering* of the gates is chosen, which ensures that gate  $g_i$  has no inputs that are outputs of a later gate  $g_j$ , where  $j > i$ . The inputs and the outputs can be ordered arbitrarily within themselves as long as the inputs are kept before the topologically ordered gates and the outputs after them. Even though the output nodes cause an overhead in Valiant's UC, they are required to fully hide the topology of the circuit in the corresponding universal circuit. If, one can observe which gates provide the output of the computation, it might reveal information about the structure of the circuit, e.g. how many times is the result of an output gate used after being calculated. We ensure by adding nodes corresponding to the outputs that the last  $v$  nodes in  $U_{u+v+k^*}$  are the ones providing the outputs. We note that our understanding of universal circuits here slightly differs from Valiant's, since he constructs  $U_{u+k^*}$  [Val76].

Therefore, after obtaining  $G^C$  a  $\Gamma_2$  edge-universal graph  $U_{u+v+k^*}$  is constructed, into which  $G^C$  is edge-embedded. Valiant shows in [Val76] how to obtain the universal circuit corresponding to  $U_{u+v+k^*}$  and how to program it according to the edge-embedding of  $G^C$ . Firstly, the first  $u$  poles become inputs, the next  $k^*$  poles are so-called universal gates, and the last  $v$  poles are outputs in the universal circuit. A *universal gate* denoted by  $U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3)$ , can compute any function with two inputs  $\text{in}_1$  and  $\text{in}_2$  and four control bits  $c_0, c_1, c_2$  and  $c_3$  as in Eq. 1.

$$\text{out}_1 = c_0 \overline{\text{in}_1 \text{in}_2} \oplus c_1 \overline{\text{in}_1} \text{in}_2 \oplus c_2 \text{in}_1 \overline{\text{in}_2} \oplus c_3 \text{in}_1 \text{in}_2. \quad (1)$$

The rest of the nodes of the edge-universal graph are translated into *universal switches* or *X gates*, denoted by  $(\text{out}_1, \text{out}_2) = X(\text{in}_1, \text{in}_2; c)$  that are defined by one control bit  $c$  and return the two input values either in the same or in reversed order as in Eq. 2.

$$\text{out}_1 = \bar{c} \text{in}_1 \oplus c \text{in}_2, \quad \text{out}_2 = c \text{in}_1 \oplus \bar{c} \text{in}_2. \quad (2)$$

The programming of the universal circuit means specifying the control bit of each universal switch and the four control bits of each universal gate. The universal gates are programmed according to the simulated gates in  $C$  and the universal switches according to the paths defined by the edge-embedding of the graph of the circuit  $G^C$  in the edge-universal graph  $U_{u+v+k^*}$ . Depending on if the path takes the same direction during the embedding (e.g. arrives from the left and continues on the left) or changes its direction at a given node (e.g. arrives from the left and continues on the right), the control bit of the universal switch can be programmed accordingly. In Sect. 4.1, we detail our concrete method for programming the universal circuit and discuss efficient implementations of universal gates and switches.

## 2.2 Universal Circuit Construction from [KS08b]

The universal circuit construction from [KS08b] is built from three main building blocks (cf. full version [KS16]) that we summarize in this section. The construction uses efficient building blocks for hiding the wiring of the  $u$  inputs and  $v$  outputs, using the fact that the maximum number of inputs to a circuit with  $k$  gates is  $2k$  and the maximum number of outputs is  $k$ . A recursive building block with size  $\mathcal{O}(k \log^2 k)$  is constructed for hiding the wiring between the gates.

For hiding the input wiring, a *selection block*  $S_{2k \geq u}^u$  is used, i.e., a programmable block that selects for  $2k$  outputs one of  $u \leq 2k$  inputs. This means that with the  $u$  inputs of circuit  $C$ , it can be programmed to assign the output wires according to the original structure of  $C$  and assign duplicates to the rest of the wires. The authors show an efficient implementation of selection blocks with size  $\mathcal{O}(k \log k)$  and depth  $\mathcal{O}(k)$  with a small constant factor [KS08b].

For hiding the output wiring, the authors use a smaller selection block. We note that the usage of their so-called *truncated permutation block* is enough to

program the output wires according to the original topology of  $C$  as no duplicates can occur. This truncated permutation block  $TP_v^{k \geq v}$  permutes a subset of the maximal  $k$  inputs to the  $v \leq k$  outputs. An efficient construction of size  $\mathcal{O}(k \log v)$  and depth  $\mathcal{O}(\log k)$  is given in [KS08b].

A *universal block*  $UB_k$  is placed between the input selection block and the output permutation block. It takes care of the simulation of the gates using universal gates and ensures that each possible wiring can be implemented in the UC. The universal block construction is recursive, makes use of two universal blocks of smaller size with a selection block and a mixing block (essentially a layer of universal switches with one output) in between them. The  $\mathcal{O}(k \log^2 k)$  size of this universal block is asymptotically not optimal and its  $\mathcal{O}(k \log k)$  depth is also a factor of  $\log k$  larger than Valiant's UC's. Thus, despite the efficiency of the other two building blocks, the construction from [KS08b] yields larger circuits than Valiant's UC in most cases. However, we note that using some of its building blocks can be beneficial in some scenarios (cf. Sect. 3.2).

### 3 The Size and the Depth of Valiant's Construction

In this section, we obtain new formulae for the size and the depth of Valiant's construction: the  $\Gamma_1$  edge-universal graph construction is described in Sect. 3.1 and the universal circuit construction in Sect. 3.2. The *size of the edge-universal graph* is the number of nodes, counting all the poles and nodes created while using Valiant's construction. The *depth of the edge-universal graph* is the number of nodes on the longest path between any two nodes. When considering UCs and the PFE application, since XOR gates can be evaluated for free in secure computation [KS08a], the *ANDsize of the universal circuit* is the number of AND gates that are needed to realize the UC in total. The *ANDdepth of the universal circuit* in this scenario is the maximum number of AND gates between any input and output. For the sake of generality, we give the *total size* and *depth* of Valiant's UC construction with respect to both the AND and XOR gates that are used. Our implementation of universal gates and switches is optimized for PFE (cf. Sect. 4.1) and therefore uses the fewest AND gates possible. However, the total size and depth can be relevant when optimizing for other applications, in which case our implementation gives an upper bound that can be improved. For instance, when XOR and AND gates have the same costs, one needs to minimize the total number of gates instead of the number of AND gates as in [LMS16].

#### 3.1 The Size and the Depth of the $\Gamma_1$ Edge-Universal Graph

In the skeleton, node  $A$  in Fig. 1a is redundant, since one can choose to embed the edge  $(y, n-1)$  as  $(p_y, p_{n-1})$  through  $Q$ , and  $(z, n)$  as  $(p_z, p_n)$  through  $R$  for any  $y$  and  $z$  nodes [Val76]. Thus, the number of nodes other than poles  $\text{EXACT}(n)$ , for even  $n$  becomes

$$\text{EXACT}(n) = 2 \cdot \text{EXACT}\left(\frac{n-2}{2}\right) + 5 \cdot \frac{n-2}{2}. \quad (3)$$

For odd  $n$ , the construction makes use of  $\frac{n-1}{2}$  poles in  $Q$  and  $\frac{n-3}{2}$  poles in  $R$ . Then, edge  $(y, n)$  is embedded as  $(p_y, p_n)$  through  $Q$  for any  $y$  node, and node  $A$  is again redundant. Thus,

$$\text{EXACT}(n) = \text{EXACT}\left(\frac{n-1}{2}\right) + \text{EXACT}\left(\frac{n-3}{2}\right) + 5 \cdot \frac{n-3}{2} + 3. \quad (4)$$

Using these recursive formulae, given the value  $n$ , it is possible to obtain the exact number of nodes other than poles in  $U_n$ . Valiant includes optimizations for starting off the recursion: for 1, 2, 3, 4, 5 and 6 nodes; the respective number of additional nodes are 0, 0, 0, 3, 7 and 9 (cf. Figs. 1c, d and e). Thus, a simple algorithm using dynamic programming based on the recursion relations of Eqs. 3 and 4 yields the exact number of nodes other than the original  $n$  poles that are created during the edge-universal graph construction. It depends on the parity of the input  $n$  at each iteration and unfortunately does not yield a closed formula for the size of Valiant's edge-universal graph construction, which is  $n + \text{EXACT}(n)$ .

Valiant states that using his method, an edge-universal graph for  $\Gamma_1(n)$  can be found “*with fewer than  $\frac{19}{8}n \log_2 n$  nodes, and fanin and fanout 2*” [Val76]. As mentioned in Sect. 2.1, we consider the more detailed algorithm that yields the result with a slightly larger prefactor of  $2.5n \log_2 n$  instead of  $2.375n \log_2 n$ . In this section, we sharpen this bound and give an approximate closed formula for the size of the construction. We first give upper and lower bounds, and then derive an approximation for a closed formula. For our lower bound, we consider the case when only the formula for even numbers, i.e., Eq. 3, is considered. This yields our lower bound of

$$n + 5 \left( \sum_{i=0}^{\log_2 n - 1} 2^i \left( \frac{n}{2^{i+1}} - \frac{2(2^{i+1} - 1)}{2^{i+1}} \right) \right) = 2.5n \log_2 n - 9n + 5 \log_2 n + 10. \quad (5)$$

The upper bound can be obtained similarly, considering the case when only the formula for odd numbers with  $5 \cdot \left(\frac{n-1}{2}\right)$  is considered

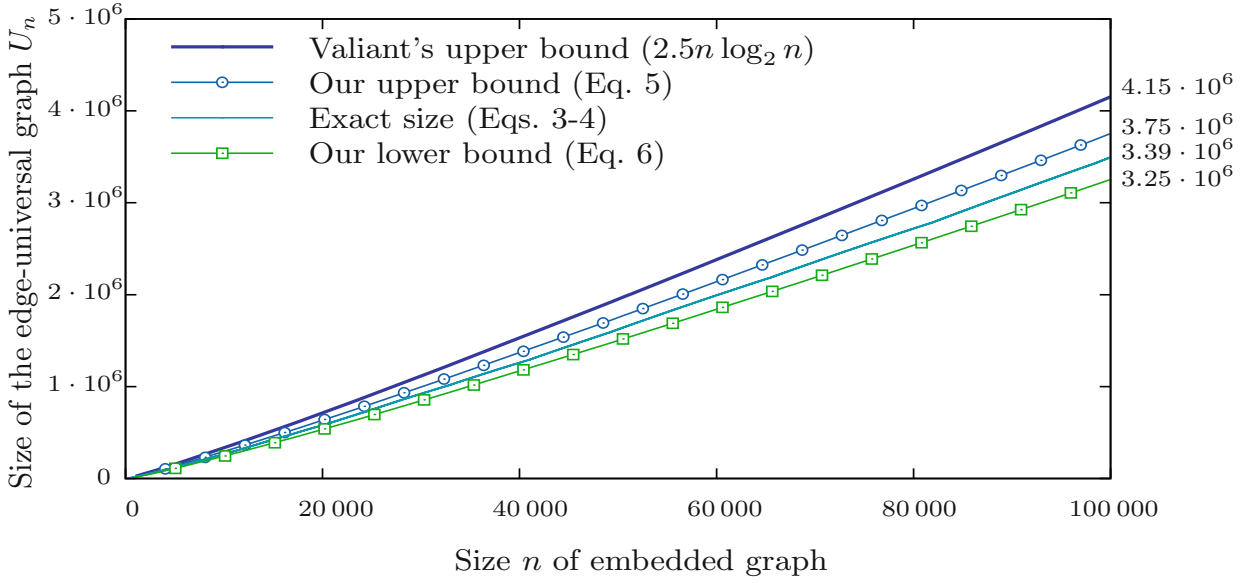
$$n + 5 \left( \sum_{i=0}^{\log_2 n - 1} 2^i \left( \frac{n}{2^{i+1}} - \frac{2^{i+1} - 1}{2^{i+1}} \right) \right) = 2.5n \log_2 n - 4n + 2.5 \log_2 n + 5. \quad (6)$$

Figure 2 depicts our upper and lower bounds along with Valiant's upper bound on the same construction for up to 100 000 nodes. We observe that the mean of our bounds is very close to the exact number of nodes. Figure 3 shows that already after a couple of hundreds of poles, it only slightly deviates from the exact number of nodes  $\text{EXACT}(n)$ . Thus, we accept

$$\text{size}(U_n) \approx 2.5n \log_2 n - 6.5n + 3.75 \log_2 n + 7.5 \quad (7)$$

as a good approximation of the closed formula for the size of the construction, noting that an estimated deviation of at most 2 % compared to the exact number of nodes, i.e.,  $\varepsilon \leq 0.02 \cdot \text{size}(U_n)$  may occur.



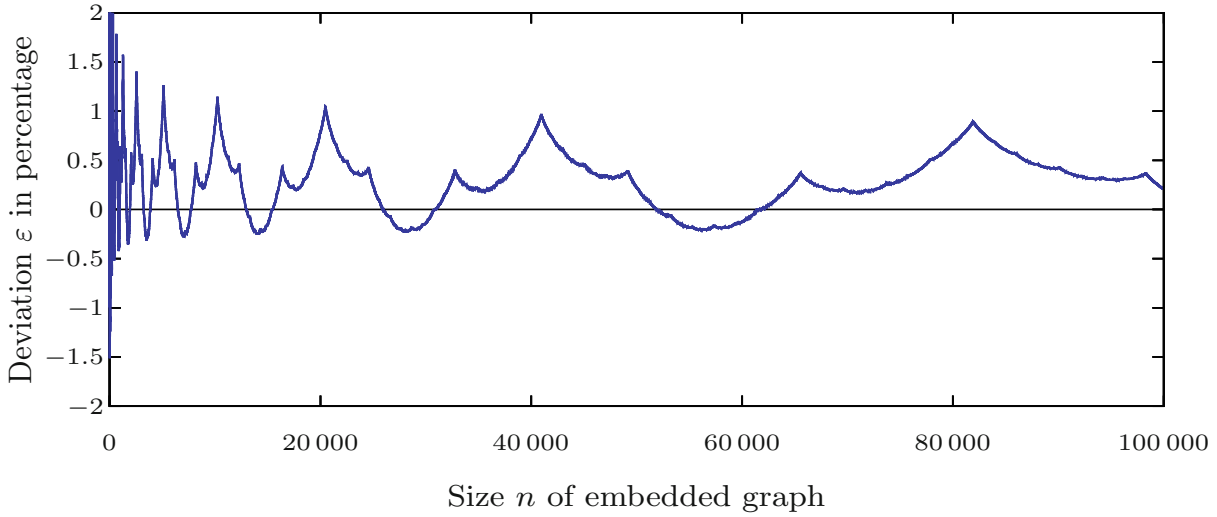


**Fig. 2.** Our upper and lower bounds for the size of Valiant's edge-universal graph construction for  $\Gamma_1(n)$  graphs, along with Valiant's upper bound on the same construction and the exact size  $\text{EXACT}(n)$ , considering the size of the embedded graph  $n \in \{1, \dots, 100\,000\}$  (Color figure online).

The depth of the edge-universal graph, i.e., the maximum number of nodes between any two nodes is defined by the number of nodes between  $p_1$  and  $p_n$  in the skeleton (cf. Figs. 1a and b). Thus,  $\text{depth}(U_n) = 3n - 3$  for even  $n$  and  $\text{depth}(U_n) = 3n - 2$  for odd  $n$ .

### 3.2 The Size and the Depth of Valiant's Universal Circuit

As described in Sect. 2.1, a universal circuit is constructed by means of an edge-universal graph for graphs with fanin and fanout 2, which is in turn constructed from two  $\Gamma_1$  edge-universal graphs with poles merged together and thus taken only once into consideration. When constructing a UC, the number of inputs  $u$ , the number of outputs  $v$  and the number of gates  $k$  is public. We set  $k^*$  as the number of gates in the equivalent fanout-2 circuit, where  $k \leq k^* \leq 2k + v$ , in order to be able to later fairly compare with the UC construction of [KS08b]. We consider  $k^*$  as the public parameter instead of  $k$ , since without the knowledge of the original number of simulated gates, it does not reveal information about the simulated circuit. If the original  $k$  is public, one can hide  $k^*$  by setting it to its maximal value  $2k + v$ . Thus, using Valiant's UC construction, a  $\Gamma_2$  edge-universal graph with  $u + v + k^*$  poles is constructed and thus, our approximative formula for the size of the  $\Gamma_2$  edge-universal graph corresponding to the graph of the circuit would become  $2 \cdot \text{size}(U_{u+v+k^*}) - (u + v + k^*)$  and the exact number would be  $u + v + k^* + 2 \cdot \text{EXACT}(u + v + k^*)$ , i.e., the  $u + v + k^*$  merged poles of the two edge-universal graphs plus the exact number of nodes other than poles. Therefore, the size of Valiant's UC is



**Fig. 3.** The deviation of the mean of our upper and lower bounds (Eqs. 5 and 6) from the exact size of the edge-universal graph  $\text{EXACT}(n) + n$ , considering the size of the embedded graph  $n \in \{1, \dots, 100\,000\}$ .

$$\begin{aligned} \text{size}(UC_{u,v,k^*}^{\text{Valiant}}) \approx & [5(u + v + k^*) \log_2(u + v + k^*) - 15(u + v + k^*) \\ & + 7.5 \log_2(u + v + k^*) + 15] \cdot \text{size}(X) + k^* \cdot \text{size}(U) \end{aligned} \quad (8)$$

and the depth stays

$$\text{depth}(UC_{u,v,k^*}^{\text{Valiant}}) \approx [2(u + v + k^*) - 2] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \quad (9)$$

When transforming the  $\Gamma_2$  edge-universal graph into a UC, the first  $u$  poles are associated with inputs, the last  $v$  poles with outputs, and the  $k^*$  poles between are realized with universal gates (cf. Eq. 1) and their programming is defined by the corresponding gates in the simulated circuit. The rest of the nodes of the edge-universal graph are translated into universal switches (cf. Eq. 2), whose programming is defined by the edge-embedding of the graph of the circuit into the  $\Gamma_2$  edge-universal graph. Thus, the size and depth of Valiant's UC can be directly derived from the size of the  $\Gamma_2$  edge-universal graph. However, we include two optimizations to obtain a smaller size of the UC. The first optimization improves already the size of the edge-universal graph and the second optimization is applied when translating the edge-universal graph into a UC description (cf. Sect. 4.1).

**1. Optimization for Input and Output Nodes:** We observe that obviously circuit inputs need no ingoing edges and circuit outputs need no outgoing edges. Therefore, since  $u, v$  and  $k^*$  are publicly known, we optimize by deleting nodes that become redundant while canceling the edges going to the first  $u$  (input) and coming from the last  $v$  (output) nodes. Depending on the parity of  $u, v$  and  $u + v + k^*$ , the number of redundant switching nodes is  $u + v - 3 \pm 1$  in both  $\Gamma_1$  edge-universal graphs that build up the graph of the UC. Therefore, we have, on average,  $2(u + v - 3)$  redundant nodes, which number we use in our calculations further on. This optimization also affects the depth by, on average,  $u + v - 3$ .

**2. Optimization for Fanin-1 Nodes:** We observe that in the skeleton of the  $\Gamma_1$  edge-universal graph construction there is a fanin-1 node (denoted with  $B$  in Figs. 1a and b). Such fanin-1 nodes exist in the base-cases for a small number of poles as well (cf. Figs. 1c, d and e). These nodes are important to achieve fanin and fanout 2 of each nodes in the graph, but can be ignored and replaced with wires when translated into a circuit description, essentially resulting in the same UC. According to Valiant's construction, these gates would translate into universal switches with one real input (and an other arbitrary one). Instead, we translate each of them into two wires and therefore set the second input to the same as the first one. Since at least one such node can be ignored in each subgraph when nodes are translated into gates, this results in altogether around

$$2 \cdot \left( \sum_{i=0}^{\log_2(u+v+k^*)-1} 2^i \right) - 1 = 2(u+v+k^*) - 3 \quad (10)$$

less gates for the two  $\Gamma_1$  edge-universal graphs. This improvement has no effect on the depth of the construction.

Since both the size and the depth are dependent on the underlying representation of the circuit building blocks (of the universal gate  $U$  and of the universal switch or  $X$  gate), and the secure computation protocol, we express the size of the universal circuit with the size and depth of  $U$  and of  $X$  as parameters. Including the above optimizations of altogether  $4(u+v) + 2k^* - 9$ , the approximate formula for the size of Valiant's optimized UC construction becomes

$$\begin{aligned} \text{size}(UC_{u,v,k^*}^{\text{opt}}) \approx & [5(u+v+k^*) \log_2(u+v+k^*) - 17k^* - 19(u+v) \\ & + 7.5 \log_2(u+v+k^*) + 24] \cdot \text{size}(X) + k^* \cdot \text{size}(U). \end{aligned} \quad (11)$$

To obtain the exact size of the UC, we use the recursive relations depicted in Eqs. 3 and 4 and include our optimizations. Thus, we obtain

$$\begin{aligned} \text{size}_{\text{exact}}(UC_{u,v,k^*}^{\text{opt}}) = & [2 \cdot \text{EXACT}(u+v+k^*) \\ & - 4(u+v) - 2k^* + 9] \cdot \text{size}(X) + k^* \cdot \text{size}(U). \end{aligned} \quad (12)$$

From the depth of the edge-universal graph, the depth of the UC becomes

$$\text{depth}(UC_{u,v,k^*}^{\text{opt}}) \approx [u+v+2k^*+3] \cdot \text{depth}(X) + k^* \cdot \text{depth}(U). \quad (13)$$

Depending on the application,  $\text{size}(X)$  and  $\text{size}(U)$  as well as  $\text{depth}(X)$  and  $\text{depth}(U)$  can be optimized. Due to the PFE application, where XOR gates can be evaluated for free, we assess the ANDsize and ANDdepth of our AND-optimized implementations of universal gates and switches (cf. Sect. 4.1). In general, a universal gate can be realized with 3 AND gates (and 6 XOR gates), and ANDdepth of 2 (total depth of 6). Universal switches can be realized with only one AND gate (and 3 XOR gates), and ANDdepth of 1 (total depth of 3) [KS08a].



For private function evaluation, the size and the depth of  $U$  can be further optimized depending on the underlying secure computation protocol. In case the SFE implementation uses Yao's garbled circuit protocol [Yao86], both  $\text{ANDsize}(U)$  and  $\text{ANDdepth}(U)$  can be minimized to 1, due to the fact that in some garbling schemes the evaluator does not learn the type of the evaluated gate such as in case of garbled 3-row-reduction [NPS99]. Therefore, a universal gate can be implemented with one 2-input non-XOR gate [PSS09].

**Optimized Hybrid Universal Circuit Construction:** We investigate if hybrid methods utilizing building blocks of both UC constructions, i.e., of both [Val76] summarized in Sect. 2.1 and [KS08b] in Sect. 2.2, could yield better size. The simulation of the  $k$  gates of the original circuit is asymptotically more efficient using Valiant's UC construction due to the logarithmic factor, despite the overhead caused by taking the equivalent fanout-2 circuit with  $k^*$  gates, where  $k \leq k^* \leq 2k + v$ . However, we calculate if the modular approach of [KS08b] using a selection block  $S_{m \geq u}^u$  for selecting the input variables or a truncated permutation block  $TP_v^{k^* \geq v}$  for the output variables results in a smaller size.

Placing a selection block on top of Valiant's UC with  $m$  universal gates would imply a selection block  $S_{m \geq u}^u$  which is then programmed to direct the  $u$  inputs of the circuit to the proper inputs of the  $m$  universal gates. Depending on how the output nodes are represented,  $m$  is either  $2(k^* + v)$  for the case when including the outputs in Valiant's construction or  $2k^*$  for the construction with a truncated permutation block. In the latter case,  $TP_v^{k^* \geq v}$  takes care of permuting a subset of the outputs of the  $k^*$  gates, resulting in the  $v$  outputs of the UC. A selection block  $S_{m \geq u}^u$  has size  $\frac{u+m}{2} \log_2 u + m \log_2 m - u + 1$  and depth  $2 \log_2 u + 2 \log_2 m + m - 2$ , and a truncated permutation block  $TP_v^{k^* \geq v}$  has size  $\frac{k^*+v}{2} \log_2 v - 2v + k^* + 1$  and depth  $\log_2 k^* + \log_2 v - 1$  [KS08b] (cf. full version [KS16]).

Let us take three scenarios into consideration, depending on the number of inputs  $u$  and the number of outputs  $v$ . The number of gates in the circuit to be simulated is  $k$  and the number of gates in the equivalent fanout-2 circuit is  $k^*$  with  $k \leq k^* \leq 2k + v$ .

1. **Constant I/O Case:**  $u = c_1$  **constant**,  $v = c_2$  **constant:** If both  $u$  and  $v$  are constant values  $c_1$  and  $c_2$  respectively, as is the case in many applications that compute a non-trivial function with relatively few inputs and outputs, the size of the selection block becomes  $\approx 2k^* \log_2 k^* + (2 + \log_2 c_1)k^*$  and the size of the truncated permutation block is  $\approx (0.5 \log_2 c_2 + 1)k^*$ . With Valiant's UC construction, the overhead caused by a constant number of inputs and outputs is around  $5(c_1 + c_2) \log_2 k^*$ . The depth of Valiant's UC is only affected with constant overhead, while the depth of the selection and permutation blocks are  $\approx 2k^* + 2 \log_2 k^*$  and  $\approx \log_2 k$ , respectively. Thus, both for the inputs and the outputs, Valiant's UC is an asymptotically better solution in the case with a constant number of inputs and outputs.

- 2. Many Inputs:**  $u \sim k$ ,  $v = c$  **constant:** For many inputs where  $u$  is around the number of gates  $k$  and we have a constant number of  $c$  outputs, we include these  $c$  nodes in Valiant's UC instead of using a truncated permutation block due to the same reasoning as in the previous case. However, a selection block can be constructed to direct  $k$  inputs to  $k^* + c$  universal gates. Thus, its size becomes  $\approx 2k^* \log_2 k^* + k^* \log_2 k + 0.5k \log_2 k + 2k^* - k + 3c \log_2 k^*$  and its depth  $\approx 2k^* + 2 \log_2 k^* + 2 \log_2 k$ . In case of Valiant's UC construction,  $k$  inputs result in an overhead of  $\approx 5k \log_2 k - 9k + 5c \log_2 k$  for the size and  $\approx k$  for the depth, since a large part (up to a half) of the circuit is built in order to hide the input wiring. Therefore, in this scenario it is often worth to use a hybrid method, utilizing the selection block from [KS08b] for input selection. Our *many inputs hybrid* construction places a selection block on top of a UC with  $k^* + c$  universal gates and has approximate size when  $u \sim k$  and  $v$  is constant  $c$

$$\begin{aligned} \text{size}(UC_{k,c,k^*}^{\text{many I}}) &\approx [7k^* \log_2 k^* + k^* \log_2 k + 0.5k \log_2 k - k - 15k^* \\ &\quad + (7.5 + 5c) \log_2 k^* + 3c \log_2 k^* + \mathcal{O}(1)] \cdot \text{size}(X) + k^* \cdot \text{size}(U) \end{aligned} \quad (14)$$

and approximate depth

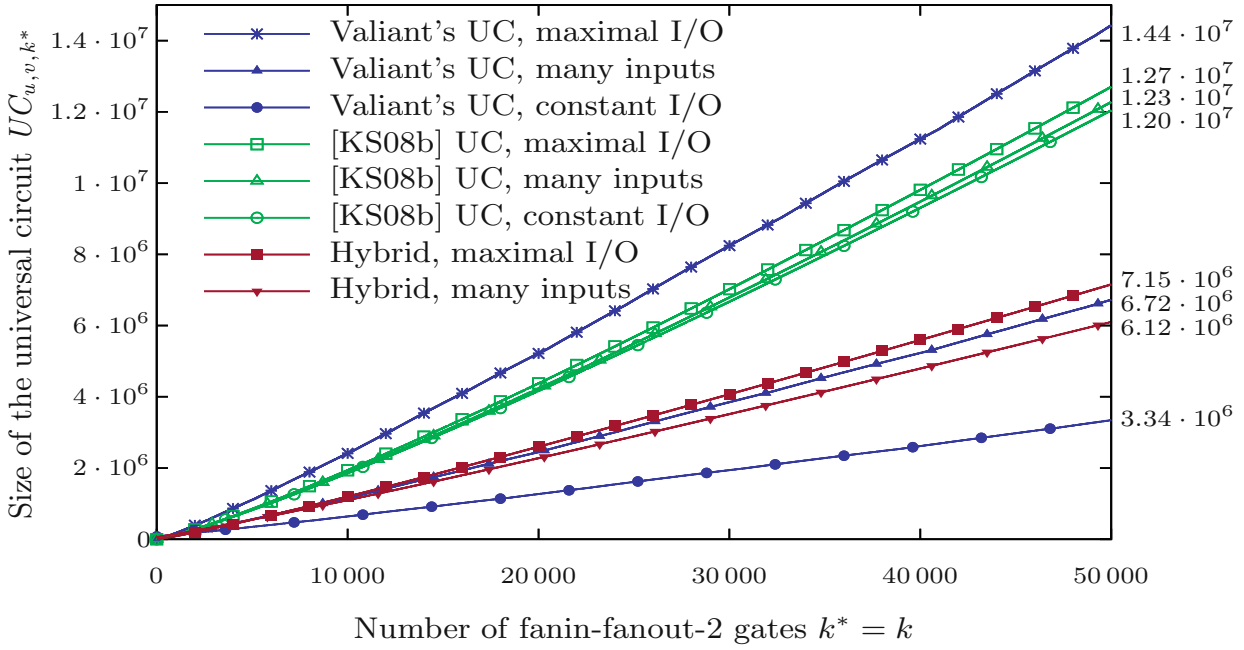
$$\begin{aligned} \text{depth}(UC_{k,c,k^*}^{\text{many I}}) &\approx [4k^* + 2 \log_2 k^* + 2 \log_2 k + \mathcal{O}(1)] \cdot \text{depth}(X) \\ &\quad + k^* \cdot \text{depth}(U). \end{aligned} \quad (15)$$

- 3. Maximal I/O Case:**  $u \sim 2k$ ,  $v \sim k$ : For circuits with  $u \sim 2k$  inputs and  $v \sim k$  outputs, we discuss the possibility of using both an input selection block and an output permutation block. The size of the selection block is  $\approx 2k^* \log_2 k^* + k^* \log_2 k + k \log_2 k + 3k^* - k$  and its depth becomes  $\approx 2k^* + 2 \log_2 k^* + 2 \log_2 k$ , which is more beneficial (when it comes to the size) than the  $\approx 10k \log_2 k - 12k$  size overhead and  $\approx 2k$  depth overhead in Valiant's construction caused by  $2k$  inputs (up to half of the UC is constructed for inputs only). The truncated permutation block has size  $\approx 0.5k^* \log_2 k + 0.5k \log_2 k + k^* - 2k$  and depth  $\approx \log_2 k^* + \log_2 k$ , while the same amount of outputs in Valiant's construction introduces at least  $5k \log_2 k - 9k$  new switches with depth of  $\approx k$ . Thus, for the case when the maximal  $2k$  inputs and  $k$  outputs are considered, we conclude that it is advantageous to use our *maximal I/O hybrid* construction, utilizing Valiant's graph construction for the  $k^*$  gates [Val76], a selection block for the inputs and a truncated permutation block for the outputs [KS08b]. This yields an approximate size when  $u \sim 2k$  and  $v \sim k$

$$\begin{aligned} \text{size}(UC_{2k,k^*,k}^{\text{max I/O}}) &\approx [7k^* \log_2 k^* + 1.5k^* \log_2 k + 1.5k \log_2 k - 13k^* - 3k \\ &\quad + 7.5 \log_2 k^* + \mathcal{O}(1)] \cdot \text{size}(X) + k^* \cdot \text{size}(U) \end{aligned} \quad (16)$$

and an approximate depth

$$\begin{aligned} \text{depth}(UC_{2k,k^*,k}^{\text{max I/O}}) &\approx [4k^* + 3 \log_2 k^* + 3 \log_2 k + \mathcal{O}(1)] \cdot \text{depth}(X) \\ &\quad + k^* \cdot \text{depth}(U). \end{aligned} \quad (17)$$



**Fig. 4.** Comparison between the sizes of the universal circuit constructions for  $k^* = k \in \{0, \dots, 50\,000\}$  gates, considering the three scenarios: *constant I/O* with constant number of inputs and outputs, *many inputs* with  $\sim k$  inputs and constant outputs and *maximal I/O* with  $\sim 2k$  inputs and  $\sim k$  outputs (Color figure online).

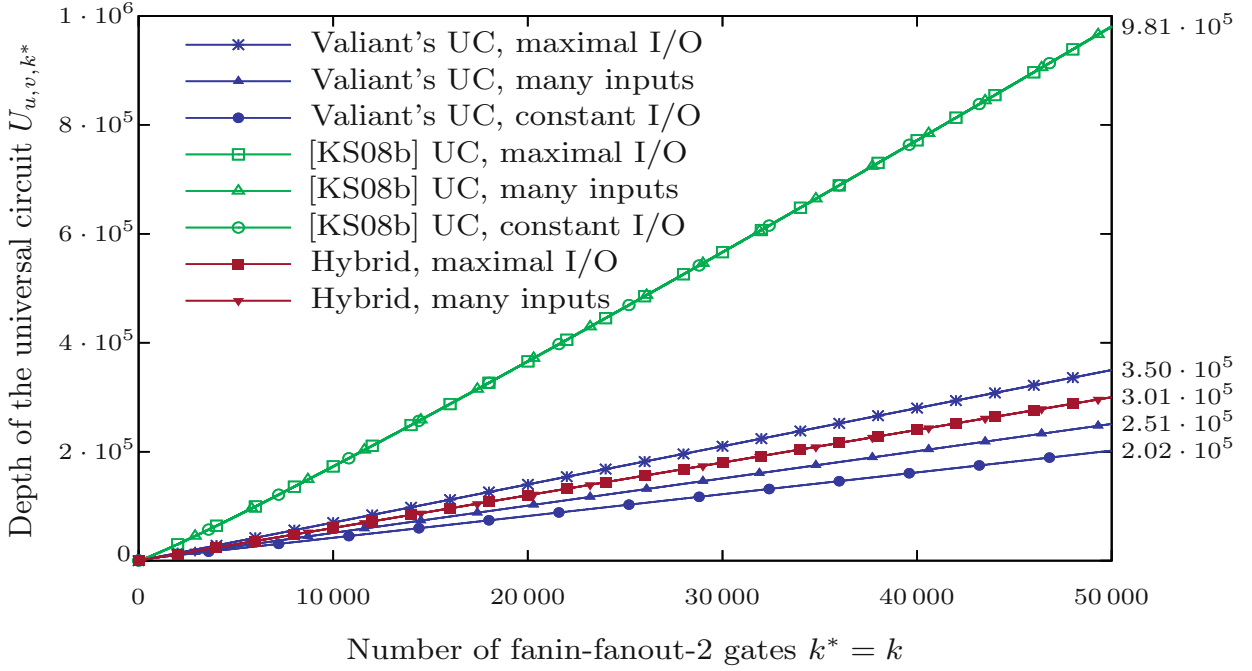
We conclude that in case of a large number of inputs and outputs it is beneficial to construct a hybrid UC, making use of both existing constructions (cf. Sects. 2.1 and 2.2). Most practical applications have input and output with constant size and only some specific applications use input size linear in the number of gates (e.g. simple computations on large databases). Thus, we consider Valiant's construction as the most beneficial for general purposes, however we have shown, that one can optimize the construction for many inputs or outputs by adding selection or truncated permutation blocks from [KS08b].

**Comparison with the Universal Circuit Construction from [KS08b].** In [KS08b], a universal circuit construction was proposed with approximate size  $1.5k \log_2^2 k + 2.5k \log_2 k$ . This was calculated with the doubled size of the universal switches, not yet considering the free-XOR optimizations of [KS08a]. We recalculated the size of the construction with our additional optimization for the outputs described in Sect. 2.2. We give our detailed calculations in the full version [KS16] and summarize its exact size here as

$$\begin{aligned} \text{size}(UC_{u,v,k}^{[\text{KS08b}]}) = & [0.75k \log_2^2 k + 2.25k \log_2 k + (0.5 + k) \log u + \\ & (0.5k + 0.5v) \log v + 5k - u - 2v] \cdot \text{size}(X) + k \cdot \text{size}(U), \end{aligned} \quad (18)$$

and from [KS08b] we know that its depth is

$$\begin{aligned} \text{depth}(UC_{u,v,k}^{[\text{KS08b}]}) = & [k \log_2 k + 2k + 7 \log_2 k + 2 \log_2 u + \\ & \log_2 v - 14] \cdot k \cdot \text{depth}(U). \end{aligned} \quad (19)$$



**Fig. 5.** Comparison between the depths of the universal circuit constructions for  $k^* = k \in \{0, \dots, 50\,000\}$  gates, considering the three scenarios: *constant I/O* with constant number of inputs and outputs, *many inputs* with  $\sim k$  inputs and constant outputs and *maximal I/O* with  $\sim 2k$  inputs and  $\sim k$  outputs (Color figure online).

It was concluded in [KS08b] that this construction outperforms Valiant's construction for circuits with up to 5 000 gates. However, this was achieved using the assumption that Valiant's UC has size  $\approx 9.5(u + 2v + 2k) \log_2(u + 2v + 2k)$ , which can vary between two to four times its actual size. On the one hand, a factor of two of this difference is due to the free-XOR optimizations in [KS08a]. On the other hand, [KS08b] used the maximal  $k^* = 2k + v$  in their approximation. In Sect. 4.2, we show on concrete example circuits that  $k^*$  stays significantly below this upper bound. The construction described in detail in Sect. 2.1 has a larger constant factor 5, but due to the logarithmic factor it outperforms the construction from [KS08b] (Sect. 2.2) already for a few hundred gates in the constant I/O case. Figures 4 and 5 compare the sizes and depth of the different UC constructions, respectively in the three scenarios described above, with the lowest possible gate number  $k^* = k$ . When considering the hybrid approach, the method corresponding to the given scenario is indeed always the most efficient construction for many inputs and/or outputs. We give a comparison for the upper bound case  $k^* = 2k + v$ , for the sizes of all universal circuit constructions for well-known circuits from [TS15] and compare their structure in the full version [KS16].

## 4 Implementing Valiant's Universal Circuit in Practice

In this section, we detail the challenges that we faced while demonstrating the practicality of Valiant's universal circuit construction. We show how to construct

a universal circuit from a standard circuit description and how to program it accordingly. We validate our results with an implementation, creating a novel toolchain for private function evaluation, using two existing frameworks as front-end and backend of our application. We emphasize that our tool for constructing and programming UC is generic and can easily be adapted to other secure computation frameworks or other applications of UCs listed in Sect. 1.2.

#### 4.1 Our Tool for Universal Circuit Construction and Toolchain for Private Function Evaluation

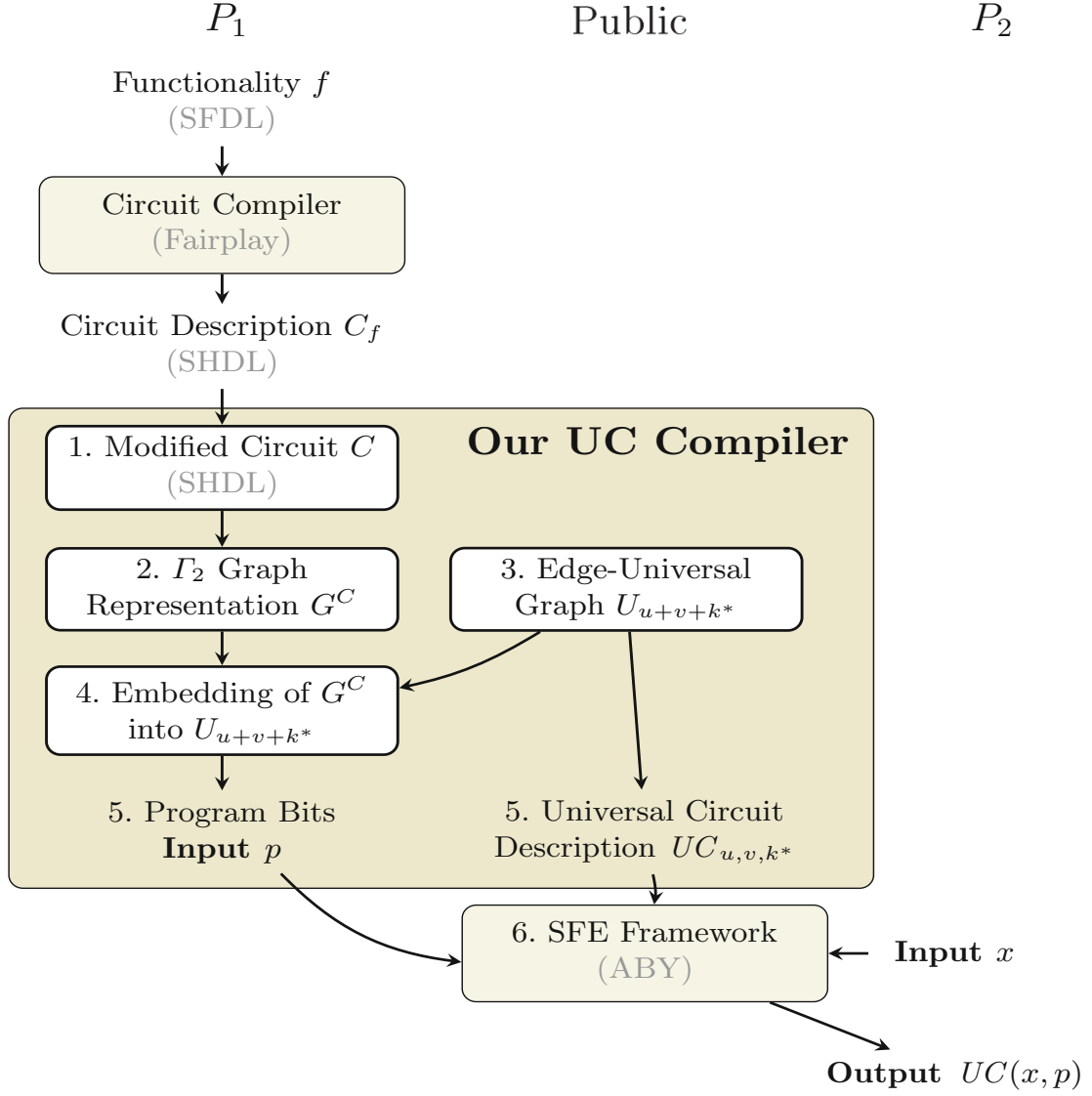
The architecture of our toolchain for PFE using universal circuits is shown in Fig. 6. In this section, we describe its different artifacts and its use of the Fairplay [MNPS04] and ABY [DSZ15] frameworks. Our implementation is available online at <http://encrypto.de/code/UC>.

**Step 1. Compiling Input Circuits from High-Level Functionality:**

Due to its easy adoptability, we decided to use the Fairplay compiler [MNPS04, BNP08] with the FairplayPF extension [KS08b] to translate the functionality described in the high-level SFDL format to the Fairplay circuit description called Secure Hardware Definition Language (SHDL). The FairplayPF extension already converts circuits with gates of an arbitrary fanin into gates with at most two inputs, which is required for Valiant’s construction as well. However, in case of Valiant’s UC construction, there is another restriction on the input circuit. It has to have fanout 2, i.e., the outputs of all the gates and inputs can only be used as the input of at most two later gates.

In case the input circuit does not follow this restriction, an algorithm places a binary tree in place of each gate with fanout larger than 2, following Valiant’s proposition: “Any gate with fanout  $x + 2$  can be replaced by a binary fanout tree with  $x + 1$  gates” [Val76, Corollary 3.1]. This is done using so-called *copy gates*, i.e., identity gates, each of them eliminating one from the extra fanout of the original gate. An upper bound can be given on the number of copy gates. The class of Boolean functions with  $u$  inputs and  $v$  outputs that can be realized by acyclic circuits with  $k$  gates and arbitrary fanout, can also be realized with an acyclic fanout-2 circuit with  $k \leq k^* \leq 2k + v$  gates [Val76, Corollary 3.1]. We give concrete examples in Sect. 4.2 on how this conversion changes the input circuit size for practical circuits and show that in most cases, the resulting number of gates remains significantly below the upper bound  $2k + v$ .

**Step 2. Obtaining the  $\Gamma_2$  Graph of the Circuit:** From the SHDL description of a  $C$  circuit with fanin and fanout 2, the  $\Gamma_2$  graph  $G^C$  of the circuit  $C$  can be directly generated as described in Sect. 2.1: with the number of inputs  $u$ , the number of outputs  $v$  and the number of gates  $k^*$  in circuit  $C$ ,  $G^C$  has  $u + v + k^*$  nodes and the wires are represented as edges in the graph. Then, the first  $u$  nodes in the topological order correspond to the inputs, the last  $v$  nodes to the outputs and the nodes in between them to the  $k^*$  ordered gates. We note that since  $C$  had fanin and fanout 2, the resulting  $G^C$  graph is in  $\Gamma_2(u + v + k^*)$ .



**Fig. 6.** Our toolchain for universal circuits and private function evaluation.

Therefore in  $G^C$ , each node can have at most two incoming edges, one defined to be the first and the other the second. It is possible in the modified SHDL circuit description that an internal value becomes two times the first or two times the second input of gates. This is due to the fact that in the original SHDL circuit with arbitrary fanout, a value could be the input of arbitrary number of later gates. Transforming the circuit to a fanout-2 circuit by adding copy gates allows a value to be an input only two times, but the order of the inputs is fixed. Therefore, in such a case when a value is the second time the same input to a gate (i.e., first or second), besides the two inputs, the two middle bits of the function table of the gate must be reversed as well (i.e., to compute  $f(\text{in}_1, \text{in}_2)$  instead of  $f(\text{in}_2, \text{in}_1)$ ) for the correct programming of the universal circuit in Step 5.



**Step 3. Generating  $\Gamma_2$  Edge-Universal Graph  $U_n$ :** Knowing the number of input bits  $u$ , the number of gates  $k^*$  and the number of output bits  $v$  one can construct the corresponding edge-universal graph  $U_n$ , where  $n = u + v + k^*$ , with out input-output optimization from Sect. 3.2. We note that no knowledge is necessary about the topology or the gate tables in circuit  $C$  for this step. As we described in Sect. 2.1, two edge-universal graphs for  $\Gamma_1(n)$ , i.e.  $U_{n,1}$  and  $U_{n,2}$ , are merged in order to obtain an edge-universal graph for  $\Gamma_2(n)$ , such that the poles are merged and the edges coming into and going out from them become as follows: the edges in  $U_{n,1}$  will be the first input and output for each pole, the edges in  $U_{n,2}$  will be the second input and output. For efficiency reasons, we directly generate the merged edge-universal graph, i.e., an edge-universal graph for  $\Gamma_2(n)$ , with the poles as common nodes.

We include our optimization for the input and output nodes from Sect. 3.2 and Valiant's optimizations for  $n \in \{2, 3\}$ , but do not consider Valiant's optimizations for  $n \in \{4, 5, 6\}$  (cf. Figs. 1c, d, and e). These special cases lead to a specific edge-embedding for the nodes and result in linear improvement only in very rare cases. Moreover, with our second optimization from Sect. 3.2, we ignore most of the extra nodes when the graph is translated into a universal circuit description, i.e., we have for  $n = \{4, 5, 6\}$  only  $\{3, 5, 8\}$  additional nodes other than poles, respectively, in our implementation which is already an improvement over Valiant's original optimizations.

We note that the edge-universal graph (with undefined function tables and control bits for the universal switches) can be publicly generated. However, the party programming it has to either generate or receive a copy of it for programming the control bits according to the topology of the simulated circuit (i.e., to edge-embed  $G^C$  into  $U_n$ ).

**Step 4. Programming  $U_n$  According to an Arbitrary  $\Gamma_2(n)$  Graph:** The  $\Gamma_2$  graph of the circuit  $G^C$  with  $n$  nodes is partitioned into two  $\Gamma_1(n)$  graphs  $G_1^C$  and  $G_2^C$  which are embedded into the two edge-universal graphs for  $\Gamma_1(n)$  that build up  $U_n$ . Valiant proved in [Val76] that for any topologically ordered  $\Gamma_1(n)$  graph, for any  $(i, j) \in E$  and  $(k, l) \in E$  edges there exist edge-disjoint paths in  $U_n$  between the  $i^{\text{th}}$  and the  $j^{\text{th}}$  poles and between the  $k^{\text{th}}$  and the  $l^{\text{th}}$  poles. We described Valiant's method in Sect. 2.1 and here we show the algorithm that uniquely defines these paths in  $U_n$ .

For the description of our algorithm, we first define a  $\Gamma_1(n)$  supergraph, which is a  $\Gamma_1(n)$  graph with additionally a binary tree of  $\Gamma_1$  graphs of decreasing size. These  $\Gamma_1$  graphs uniquely define the embedding of the edges into  $U_n$ . When embedding an edge  $(i, j)$  of the topologically ordered graph  $G$  into the edge-universal graph, one needs to construct the supergraph of  $G$  as described in Algorithm 1 and then look at the binary tree in the supergraph. The path of the edge  $(i, j)$  defines the edge-embedding uniquely. This means that if edge  $(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1)$  is in the left subgraph of  $G$ , then it can be embedded through subgraph  $Q$  in  $U_n$ , otherwise it is in the right subgraph of  $G$  and can be embedded through subgraph  $R$  in  $U_n$ . The unique embedding happens through  $\{r_{\lceil \frac{i}{2} \rceil}, r_{\lceil \frac{j}{2} \rceil - 1}\}$  or through  $\{q_{\lceil \frac{i}{2} \rceil}, q_{\lceil \frac{j}{2} \rceil - 1}\}$ , utilizing the unique shortest path between them, through subpoles further identified by smaller subgraphs of  $G$ .

**Algorithm 1.** SUPERGRAPH( $G$ )

---

**Input:**  $\Gamma_1(n)$  graph  $G$  with set of nodes  $V = \{1, \dots, n\}$   
**Output:**  $\Gamma_1(n)$  supergraph

- 1: Create a graph  $H$  with  $\lceil \frac{n}{2} \rceil - 1$  nodes  $\triangleright H$   $\Gamma_2$  graph (with possible loops)
- 2: **if** there exist an edge  $(i, j)$  in  $G$  **and**  $\lceil \frac{j}{2} \rceil - 1 \geq \lceil \frac{i}{2} \rceil$  **then**
- 3:     Add edge  $(\lceil \frac{i}{2} \rceil, \lceil \frac{j}{2} \rceil - 1)$  in  $H$   $\triangleright$  each pair of nodes in  $G$  is one node in  $H$
- 4: **end if**
- 5: Partition  $H$  into two  $\Gamma_1$  graphs  $G_1$  of size  $\lceil \frac{n}{2} \rceil - 1$  and  $G_2$  of size  $\lfloor \frac{n}{2} \rfloor - 1$  using König's theorem as in §2.1  
 $\triangleright$  in odd case, the  $(e, \lceil \frac{n}{2} \rceil - 1)$  edge in  $H$  for arbitrary  $e$  will be added in  $G_1$
- 6: **if**  $\text{size}(G_1) \neq 0$  **then**
- 7:     SUPERGRAPH( $G_1$ )
- 8:     Store  $G_1$  as the left subgraph of  $G$
- 9: **end if**
- 10: **if**  $\text{size}(G_2) \neq 0$  **then**
- 11:     SUPERGRAPH( $G_2$ )
- 12:     Store  $G_2$  as the right subgraph of  $G$
- 13: **end if**
- 14: delete  $H$
- 15: **return**  $G$

---

When the embedding is done (cf. full version [KS16]), for defining the control bits, each node  $x$  has at most two nodes that have ingoing edges to  $x$ , one is represented as the left parent and one as the right parent of  $x$  in the edge-universal graph. The two consecutive nodes are also saved as left and right children of  $x$ . Now, when  $x$  is a switching node and we take edges  $(v, x)$  and  $(x, w)$  in the path, we save for  $x$  if parent  $v$  and child  $w$  are on the same or on the opposite side in the edge-universal graph. This defines the control bit of each universal switch in the translated universal circuit, where left and right parent and child translate to first and second input and output, respectively. We note that in order to program  $U_n$  correctly, we require that if  $x$  is the left (right) parent of  $v$  in the edge-universal graph, then  $v$  is the left (right) child of  $x$ .

**Step 5. Generating the Output Circuit Description and the Programming of the Universal Circuit:** After embedding the graph of the simulated circuit into the edge-universal graph  $U_n$ , we write the resulting circuit in a file using our own circuit description. In the edge-universal graph, each node stores the program bit resulting from the edge-embedding (control bit  $c$  of the corresponding universal switch in Eq. 2) and each pole stores four bits corresponding to the simulated circuit (the four control bits of the function table,  $c_0, c_1, c_2, c_3$  in Eq. 1, their order possibly changed in Step 2). Thus, after topologically ordering  $U_n$ , one can directly write out the gate identifiers into a circuit file and the program bits to a programming file.

Our circuit description format starts with enumerating the inputs and ends with enumerating the outputs. We have universal gates denoted by  $U$ , universal switches denoted by  $X$  or  $Y$  depending on the number of outputs ( $X$  with two



outputs and  $Y$  with one). We note that we replace any gates that have only one input by wires in the UC, thus achieving our fanin-1 node optimization from Sect. 3.2. The wires are represented in the following manner:

$$\begin{array}{llll}
 U & \text{in}_1 & \text{in}_2 & \text{out}_1 \\
 X & \text{in}_1 & \text{in}_2 & \text{out}_1 \quad \text{out}_2 \\
 Y & \text{in}_1 & \text{in}_2 & \text{out}_1
 \end{array} \tag{20}$$

denotes that wire  $\text{out}_1$  (and possibly  $\text{out}_2$ ) is coming from a gate with input wires  $\text{in}_1$  and  $\text{in}_2$ . The program bits are not represented in the circuit format, but in a separate file, for each universal gate we save a four-bit number representing the control bits and for each universal switch we store the control bit. The output nodes are outputs of  $Y$  universal switches and are marked in the end of the file as  $O \quad o_1 \quad o_2 \quad \dots \quad o_v$ . The circuit and its programming are given in plain text files.

**Step 6. Evaluating Universal Circuits for PFE in ABY:** As an example application of UCs, we implement PFE using SFE of a universal circuit. We adapted the ABY secure two-party computation framework [DSZ15] for this purpose. Firstly, since ABY uses the free-XOR optimization from [KS08a], we construct universal gates and switches with low ANDsize and ANDdepth given in Sect. 3.2. With the cost metric we consider,  $X$  and  $Y$  gates have the same AND complexity, optimized in [KS08a] and are obtained as

$$\begin{aligned}
 \text{out}_1 &= Y(\text{in}_1, \text{in}_2; c) = (\text{in}_1 \oplus \text{in}_2)c \oplus \text{in}_1 \\
 (\text{out}_1, \text{out}_2) &= X(\text{in}_1, \text{in}_2; c) = (e \oplus \text{in}_1, e \oplus \text{in}_2) \text{ with } e = (\text{in}_1 \oplus \text{in}_2)c
 \end{aligned} \tag{21}$$

with ANDsize and ANDdepth of 1 for both universal switches.  $X$  gates are realized with one additional XOR gate compared to  $Y$  gates.

Our efficient implementation of generic universal gates uses  $Y$  gates yielding

$$\text{out}_1 = U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3) = Y[Y(c_0, c_1; \text{in}_2), Y(c_2, c_3; \text{in}_2); \text{in}_1] \tag{22}$$

with  $\text{ANDsize}(U) = 3$  and  $\text{ANDdepth}(U) = 2$ . This universal gate implementation is generic and works in all secure computation protocols. However, for Yao's garbled circuits protocol, one can further optimize it to  $\text{ANDsize}(U) = \text{ANDdepth}(U) = 1$ , as in some garbling schemes such as the garbled 3-row-reduction [NPS99] the gate being evaluated remains oblivious to the evaluator.

After constructing the efficient building blocks, the output circuit file of our UC compiler is parsed, a circuit is generated accordingly and programmed with the input program bits. We conclude that our toolchain is the first implementation of Valiant's size-optimized universal circuit that supports efficient private function evaluation.

## 4.2 Comparison of Our PFE-Toolchain with Other PFE Protocols

Mohassel et al. in [MS13] design a generic framework for PFE and apply it to three different scenarios: to the  $m$ -party GMW protocol [GMW87], to Yao's

**Table 1.** The number of symmetric-key operations using different PFE protocols: Valiant's UC with SFE, the universal circuit construction from [KS08b] or Mohassel et al.'s OT-based method from [MS13].  $u, v$  and  $k$  denote the number of inputs, outputs and gates in the simulated circuit, and  $k^*$  denotes the number of gates in the equivalent fanout-2 circuit.

Circuit	$u$	$k$	$v$	$k^* - k (\frac{k^*}{k})$	Valiant	[KS08b]	OT-based [MS13]
AES-non-exp	256	31 924	128	14 539 (1.46)	$1.150 \cdot 10^7$	$2.797 \cdot 10^7$	$6.243 \cdot 10^6$
AES-expanded	1 536	25 765	128	11 089 (1.43)	$9.211 \cdot 10^6$	$2.206 \cdot 10^7$	$4.943 \cdot 10^6$
DES-non-exp	128	19 464	64	12 290 (1.63)	$7.502 \cdot 10^6$	$1.560 \cdot 10^7$	$3.639 \cdot 10^6$
md5	512	43 234	128	22 623 (1.52)	$1.700 \cdot 10^7$	$3.995 \cdot 10^7$	$8.681 \cdot 10^6$
add_32	64	187	33	58 (1.31)	35 512	55 341	19 939
comp_32	64	150	1	1 (1.01)	19 384	40 222	15 424
mult_32x32	64	6 995	64	5 079 (1.73)	$2.522 \cdot 10^6$	$4.647 \cdot 10^6$	$1.184 \cdot 10^6$
Branching_18	72	121	4	3 (1.02)	17 312	30 994	11 994
CreditCheck	25	50	1	6 (1.12)	5 056	9 348	4 198
MobileCode	80	64	16	0 (1.00)	12 528	13 727	5 644

garbled circuits [Yao86] and to arithmetic circuits using homomorphic encryption [CDN01]. Both the two-party version of their framework with the GMW protocol and the solution with Yao's garbled circuit protocol has two alternatives: using homomorphic encryption they achieve linear complexity  $\mathcal{O}(k)$  in the circuit size  $k$  and when using a solution solely based on oblivious transfers (OTs), they obtain a construction with  $\mathcal{O}(k \log k)$  symmetric-key operations. The OT-based construction in both cases is more desirable in practice, since using OT extension the number of public-key operations can be reduced significantly [IKNP03, ALSZ13].

Since the asymptotical complexity of this construction and using Valiant's UC for PFE is the same, we compare these methods for PFE. We revisit the formulas provided in [MS13] for the PFE protocol based on Yao's garbled circuits and elaborate on the number of symmetric-key operations when the different PFE protocols are used. Mohassel et al. show that the total number of switches in their framework is  $4k \log_2(2k) + 1$  that are evaluated using OT extension, for which they calculate  $8k \log_2(2k) + 8$  symmetric-key operations together with  $5k$  operations for evaluating the universal gates with Yao's protocol. We count only the work of the party that performs most of the work, i.e.,  $4k$  symmetric-key operations for creating a garbled circuit with  $k$  gates and 3 symmetric-key operations (two calls to a hash function and one call to a pseudorandom function (PRF)) for each OT using today's most efficient OT extension of [ALSZ13]. Hence, according to our estimations, the protocol of [MS13] requires  $12 \log_2(2k) + 4k + 12$  symmetric-key operations.

In the same way, we assume that in our case, for evaluating both the universal gates and switches, the garbler needs  $4k$  symmetric-key operations. Thus, for a fair comparison, we essentially update Table 4 from the full version of [MS13, Appendix J.1], where Valiant’s UC size was calculated with assumed  $k^* = 2k + v$ , without calculating 4 operations for the garbling.

We took our example circuit files of varying size in Table 1 from two different sources and elaborate on the resulting number of symmetric-key operations using the different constructions. The first 7 circuits we obtained from the function set of [TS15] and the last three from the FairplayPF extension of the Fairplay compiler [MNPS04, KS08b]. The example circuits that we took from [TS15] had to be converted to our desired SHDL format, which was a necessary step in order to be able to elaborate on the performance of these more complicated circuits as well. We included the INV gates in the function table of the consecutive gate and therefore, resulted in smaller gate numbers  $k$  for the equivalent SHDL circuits with arbitrary fanout. Then, these SHDL circuits were considered as input circuits for our tool.

We now compare the size of the three two-party PFE protocols: the two UC-based PFE with secure computation and the OT-based method of [MS13]. We assess our findings in Table 1. We note that our numbers are estimations, i.e., we do not consider that [MS13] works with circuits made up solely of NAND gates. Since Valiant’s UC construction depends also on the number of gates with fanout more than 2 in the original circuit, we include the number of copy gates,  $(k^* - k)$  in the table. We emphasize the ratio between the new number of gates  $k^*$  and the original number of gates  $k$  and conclude that in general circuits, it is well below the maximal  $\frac{k^*}{k} \sim 2$ . The size of the UC construction from [KS08b] obviously makes their method less efficient, in our examples using more than twice as many symmetric-key operations as the method with Valiant’s UC and four times as many as Mohassel et al.’s efficient OT-based method [MS13]. We conclude that universal circuits are not the most efficient solution to perform PFE, however, we show the feasibility of generating and evaluating UCs simulating large circuits. We emphasize that even though the PFE-specific protocol from [MS13] achieves better results for PFE, universal circuits are generic and can be applied for various other scenarios (cf. Sect. 1.2), and the most efficient UC construction is Valiant’s construction.

**Our Experimental Results.** We validated the practicality of Valiant’s universal circuit construction with an efficient implementation. We ran our experiments on two Desktop PCs, each equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, that are connected via Gigabit-LAN and give our benchmarks in Table 2. We are able to generate UCs up to around 300 000 gates of the simulated circuit, i.e., which results in billions of gates in the UC. Until now, the only implementation of universal circuits was given in [KS08b], which is outperformed by Valiant’s construction already for a couple of hundred gates (cf. Figs. 4 and 5) due to its asymptotically larger complexity. We show the real practicality of UCs through experimental results proving the efficiency of our

**Table 2.** Running time and communication for our UC-based PFE implementation with ABY. We include the compile time, the I/O time of the UC compiler, and the evaluation time (in milliseconds) and the total communication (in Kilobytes) between the parties in GMW as well as in Yao sharing.

Circuit	UC Compile Time (ms)	UC I/O Time (ms)	GMW		Yao	
			Time (ms)	Communic. (KB)	Time (ms)	Communic. (KB)
AES-non-exp	2 909.2	6 331.2	5 522.08	137 561.13	2 349.35	88 417.61
AES-expanded	2 103.7	5 063.6	4 136.72	109 033.79	1 878.75	70 097.48
DES-non-exp	1 596.2	4 173.5	2 695.51	76 644.38	1 310.52	48 180.69
md5	4 043.5	8 785.4	7 041.12	169 558.83	3 547.68	110 043.59
add_32	11.4	63.8	31.97	457.77	26.49	224.77
comp_32	5.8	34.1	29.94	340.23	8.90	159.73
mult_32x32	328.9	1 443.2	1 092.46	31 053.53	539.98	18 741.85
Branching_18	4.8	31.4	26.23	307.77	17.34	145.87
CreditCheck	1.2	11.4	26.25	113.35	5.67	45.15
MobileCode	3.2	26.3	25.71	202.50	28.16	103.45

implementation of PFE with the ABY framework [DSZ15]. Furthermore, due to its asymptotically smaller depth, we are also able to evaluate our generated UCs with the GMW protocol [GMW87], whereas the construction from [KS08b] was only evaluated with Yao’s garbled circuit protocol. We do not directly compare our runtimes with the method of [MS13], since to the best of our knowledge, their framework has not yet been implemented.

Converting from circuit descriptions and writing into and reading out from files slows down the program significantly, but it still achieves good performance for practical circuits such as AES and DES. Our implementation in ABY can evaluate most of the circuits in both the GMW and Yao’s protocols, but for some examples it runs out of memory (e.g. SHA-256). However, improvements on SFE protocols imply improvements on UC-based PFE frameworks as well. As can be seen in Table 2, the evaluation time and the communication in case of Yao’s garbled circuit protocol is about a factor of two smaller than that of the GMW protocol. This difference is due to the more efficient universal gate construction with only one gate for the case of Yao’s protocol in contrast to the universal gates used in the GMW protocol with  $\text{ANDsize} = 3$  and  $\text{ANDdepth} = 2$ .

**Acknowledgements.** This work has been co-funded by the European Union’s 7th Framework Program (FP7/2007–2013) under grant agreement n. 609611 (PRAC-TICE), by the German Federal Ministry of Education and Research (BMBF) within CRISP, by the DFG as part of project E3 within the CRC 1119 CROSSING, and by the Hessian LOEWE excellence initiative within CASED. We thank Michael Zohner and Daniel Demmler for helping with the implementation in ABY, and the anonymous reviewers of Eurocrypt 2016 for their helpful comments on our paper.

## References

- [AF90] Abadi, M., Feigenbaum, J.: Secure circuit evaluation. *J. Cryptol.* **2**(1), 1–12 (1990)
- [ALSZ13] Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: *ACM CCS 2013*, pp. 535–548. ACM (2013)
- [Att14] Attrapadung, N.: Fully secure and succinct attribute based encryption for circuits from multi-linear maps. *IACR Cryptology ePrint Archive* 2014:772 (2014)
- [BFK+09] Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) *ESORICS 2009*. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009)
- [BNP08] Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: *ACM CCS 2008*, pp. 257–266. ACM (2008)
- [BPSW07] Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: *ACM CCS 2007*, pp. 498–507. ACM (2007)
- [CCKM00] Cachin, C., Camenisch, J.L., Kilian, J., Müller, J.: One-round secure computation and secure autonomous mobile agents. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) *ICALP 2000*. LNCS, vol. 1853, p. 512. Springer, Heidelberg (2000)
- [CDN01] Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) *EUROCRYPT 2001*. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)
- [CH85] Cook, S.A., Hoover, H.J.: A depth-universal circuit. *SIAM J. Comput.* **14**(4), 833–839 (1985)
- [DDKZ13] Durnoga, K., Dziembowski, S., Kazana, T., Zajac, M.: One-time programs with limited memory. In: Lin, D., Xu, S., Yung, M. (eds.) *Inscrypt 2013*. LNCS, vol. 8567, pp. 377–394. Springer, Heidelberg (2013)
- [DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: *Network and Distributed System Security (NDSS 2015)*. The Internet Society (2015). <http://crypto.de/code/ABY>
- [FAL06] Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. *IEEE Trans. Comput.* **55**(10), 1259–1270 (2006)
- [FAZ05] Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: *ACM Electronic Commerce (EC 2005)*, pp. 147–154. ACM (2005)
- [FGP14] Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: *ACM CCS 2014*, pp. 844–855. ACM (2014)
- [FLA06] Frikken, K.B., Li, J., Atallah, M.J.: Trust negotiation with hidden credentials, hidden policies, and policy cycles. In: *Network and Distributed System Security (NDSS 2006)*, pp. 157–172. The Internet Society (2006)
- [FVK+15] Fisch, B., Vo, B., Krell, F., Kumarasubramanian, A., Kolesnikov, V., Malkin, T., Bellare, S.M.: Malicious-client security in Blind Seer: a scalable private DBMS. In: *IEEE Symposium on Security and Privacy (S&P 2015)*, pp. 395–410. IEEE (2015)
- [GGHZ14] Garg, S., Gentry, C., Halevi, S., Zhandry, M.: Fully secure attribute based encryption from multilinear maps. *IACR Cryptology ePrint Archive* 2014:622 (2014)



- [GGPR13] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013)
- [GHV10] Gentry, C., Halevi, S., Vaikuntanathan, V.: *i*-hop homomorphic encryption and rerandomizable Yao circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010)
- [GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: ACM Symposium on Theory of Computing (STOC 1987), pp. 218–229. ACM (1987)
- [IKNP03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
- [KM11] Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011)
- [KS08a] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
- [KS08b] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008)
- [KS16] Kiss, Á., Schneider, T.: Valiant's universal circuit is practical. Cryptology ePrint Archive, Report 2016/093 (2016). <http://eprint.iacr.org/2016/093>
- [LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.: Valiant's universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016). <http://ia.cr/2016/017>
- [LP09a] Lindell, Y., Pinkas, B.: A proof of security of Yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009)
- [LP09b] Lovász, L., Plummer, M.D.: Matching Theory. AMS Chelsea Publishing Series. American Mathematical Soc., Providence (2009)
- [MNPS04] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security Symposium, USENIX 2004, pp. 287–302 (2004)
- [MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013)
- [MSS14] Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014)
- [NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Electronic Commerce (EC 1999), pp. 129–139 (1999)
- [OI05] Ostrovsky, R., Skeith III, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005)
- [Pin02] Pinkas, B.: Cryptographic techniques for privacy-preserving data mining. SIGKDD Explor. **4**(2), 12–19 (2002)

- [PKV+14] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Choi, S.G., George, W., Keromytis, A.D., Bellovin, S., Seer, B.: A scalable private DBMS. In: IEEE Symposium on Security and Privacy (S&P 2014), pp. 359–374. IEEE (2014)
- [PSS09] Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009)
- [Sch08] Schneider, T.: Practical secure function evaluation. Master’s thesis, University Erlangen-Nürnberg, Germany, 27 February 2008
- [SS08] Sadeghi, A.-R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2009)
- [SY99] Sander, T., Young, A.L., Yung, M.: Non-interactive cryptocomputing for NC<sup>1</sup>. In: Foundations of Computer Science (FOCS 1999), pp. 554–567. IEEE (1999)
- [TS15] Tillich, S., Smart, N.: Circuits of basic functions suitable for MPC and FHE (2015). <http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/>
- [Val76] Valiant, L.G.: Universal circuits (preliminary report). In: ACM Symposium on Theory of Computing (STOC 1976), pp. 196–203. ACM (1976)
- [Weg87] Wegener, I.: The Complexity of Boolean Functions. Wiley, New York (1987)
- [Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: Foundations of Computer Science (FOCS 1986), pp. 162–167. IEEE (1986)

## B More Efficient Universal Circuit Constructions (ASIACRYPT'17)

---

- [GKS17] D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**More Efficient Universal Circuit Constructions**”. In: *23. Advances in Cryptology – ASIACRYPT’17*. Vol. 10625. LNCS. Full version: <https://ia.cr/2017/798>. Code: <https://encrypto.de/code/UC>. Springer, 2017, pp. 443–470. CORE Rank A. Appendix B.

[https://doi.org/10.1007/978-3-319-70697-9\\_16](https://doi.org/10.1007/978-3-319-70697-9_16)



# More Efficient Universal Circuit Constructions

Daniel Günther, Ágnes Kiss<sup>(✉)</sup>, and Thomas Schneider

TU Darmstadt, Darmstadt, Germany

guenther@ranger.de, {agnes.kiss,thomas.schneider}@crisp-da.de

**Abstract.** A universal circuit (UC) can be programmed to simulate any circuit up to a given size  $n$  by specifying its program bits. UCs have several applications, including private function evaluation (PFE). The asymptotical lower bound for the size of a UC is proven to be  $\Omega(n \log n)$ . In fact, Valiant (STOC'76) provided two theoretical UC constructions using so-called 2-way and 4-way constructions, with sizes  $5n \log_2 n$  and  $4.75n \log_2 n$ , respectively. The 2-way UC has recently been brought into practice in concurrent and independent results by Kiss and Schneider (EUROCRYPT'16) and Lipmaa et al. (Eprint 2016/017). Moreover, the latter work generalized Valiant's construction to any  $k$ -way UC.

In this paper, we revisit Valiant's UC constructions and the recent results, and provide a modular and generic embedding algorithm for any  $k$ -way UC. Furthermore, we discuss the possibility for a more efficient UC based on a 3-way recursive strategy. We show with a counterexample that even though it is a promising approach, the 3-way UC does not yield an asymptotically better result than the 4-way UC. We propose a hybrid approach that combines the 2-way with the 4-way UC in order to minimize the size of the resulting UC. We elaborate on the concrete size of all discussed UC constructions and show that our hybrid UC yields on average 3.65% improvement in size over the 2-way UC. We implement the 4-way UC in a modular manner based on our proposed embedding algorithm, and show that our methods for programming the UC can be generalized for any  $k$ -way construction.

**Keywords:** Universal circuit · Private function evaluation · Function hiding

## 1 Introduction

Universal circuits (UCs) are Boolean circuits that can be programmed to simulate any Boolean function  $f(x)$  up to a given size by specifying a set of program bits  $p_f$ . The UC then receives these program bits as input besides the input  $x$  to the functionality, and computes the result as  $UC(x, p_f) = f(x)$ . This means that the same UC can evaluate multiple Boolean circuits, only the different program bits are to be specified.

Valiant proposed an asymptotically size-optimal construction in [Val76] with size  $\Theta(n \log n)$  and depth  $\mathcal{O}(n)$ , where  $n$  is the size of the simulated Boolean circuit description of  $f(x)$ . He provides two constructions, based on 2-way and

4-way recursive structures. Recently, optimizations of Valiant’s size-optimized construction appeared in concurrent and independent works of [KS16] and [LMS16]. Both works implement Valiant’s 2-way recursive construction.

### 1.1 Applications of Universal Circuits

Size-optimized universal circuits have many applications. We review some of them here and refer to [KS16, LMS16] for further details.

**Private Function Evaluation (PFE).** Secure two-party computation or secure function evaluation (SFE) provides interactive protocols for evaluating a public function  $f(x, y)$  on two parties’ private inputs  $x$  and  $y$ . However, in some scenarios, the function  $f$  is a secret input of one of the parties. This setting is called private function evaluation (PFE). PFE of  $f(x)$  can be achieved by running SFE of  $UC(x, p_f)$ , where the UC is a public function and the program bits  $p_f$  – and therefore  $f$  – are kept private due to the properties of SFE. Protocols designed especially for PFE such as [MS13, BBKL17] achieve the same asymptotic complexity  $\mathcal{O}(n \log n)$  as PFE using UCs, where  $n$  is the size of the function  $f$ .<sup>1</sup> However, to the best of our knowledge, they have not yet been implemented, and they are not as generally applicable as PFE with UCs.

UC-based PFE can be easily integrated into any SFE framework and can directly benefit from recent optimizations. For instance, *outsourcing UC-based PFE* is directly possible with outsourced SFE [KR11]. The non-interactive secure computation protocol of [AMPR14] can also be generalized to obtain a *non-interactive PFE* protocol [LMS16].

One of the first applications for PFE was *privacy-preserving checking for credit worthiness* [FAZ05], where not only the loanee’s data, but also the loaner’s function needs to be kept private. PFE allows for running *proprietary software* on private data, such as privacy-preserving software diagnosis [BPSW07], medical programs [BFK+09], or privacy-preserving intrusion detection [NSMS14]. UCs can be applied to obviously *filter remote streaming data* [OI05] and for hiding queries in *private database management systems* such as Blind Seer [PKV+14, FVK+15].

**Applications Beyond PFE.** Universal circuits can be applied for program obfuscation. Candidates for *indistinguishability obfuscation* are constructed using a UC as a building block in [GGH+13a, BOKP15], which can be improved using Valiant’s UC implementation [KS16]. *Direct program obfuscation* was proposed in [Zim15], where the circuit is a secret key to a UC. [LMS16] mentions that UCs can be applied for secure two-party computation in the batch execution setting [HKK+14, LR15]. It can be applied for *verifiable computation* [FGP14], and

<sup>1</sup> There also exist PFE protocols with linear complexity  $\mathcal{O}(n)$  which are based on public-key primitives [KM11, MS13, MSS14]. However, the concrete complexity of these protocols is worse than that of the protocols based on (mostly) symmetric-key primitives, i.e., the OT-based PFE protocols of [MS13, BBKL17] or PFE using UCs.

for multi-hop homomorphic encryption [GHV10]. Ciphertext-policy *Attribute-Based Encryption* was proposed in [Att14], where the policy circuit is hidden [GGH+13b].

## 1.2 Related Work on Universal Circuits

Valiant defined universal circuits in [Val76] and gave two size-optimized constructions. The constructions are based on so-called edge-universal graphs (EUGs) and utilize either a 2-way or a 4-way recursive structure, also called 2-way or 4-way UCs. Both achieve the asymptotically optimal size  $\Theta(n \log n)$  [Val76, Weg87], where  $n$  is the size of the simulated circuit. The concrete complexity of the 4-way UC is  $\sim 4.75n \log_2 n$  which is smaller than that of the 2-way UC of  $\sim 5n \log_2 n$  [Val76].

The first modular UC construction was proposed by Kolesnikov and Schneider in [KS08b]. This construction achieves a non-optimal asymptotic complexity of  $\mathcal{O}(n \log^2 n)$ , and was the first implementation of UCs. A generalization of UCs for  $n$ -input gates was given in [SS08].

Recently, two independent works have optimized and implemented Valiant’s 2-way UC [KS16, LMS16]. Kiss and Schneider in [KS16] mainly focus on the most prominent application of UCs, i.e., private function evaluation (PFE). Due to the free-XOR optimization of [KS08a] in the SFE setting, they optimize the size of the UC for the number of AND gates in the resulting UC implementation and provide a framework for PFE using UCs as public function. They also propose hybrid constructions for circuits with a large number of inputs and outputs, utilizing efficient building blocks from [KS08b]. Lipmaa et al. in [LMS16] also provide an (unpublished) implementation of the 2-way UC. While keeping the number of AND gates minimal, they additionally optimize for the total number of gates, i.e., include optimizations to also reduce the number of XOR gates. They adapt the construction to arithmetic circuits and generalize the design to a  $k$ -way construction in a modular manner, for  $k \geq 2$ .

Both papers utilize 2-coloring of the underlying graphs for defining the program bits  $p_f$  for any given functionality  $f$ . Generally, 2-coloring can be utilized for any  $2^i$ -way construction. [LMS16] calculate the optimal value for  $k$  to be 3.147, and conclude that the two candidates for the most efficient  $2^i$ -way constructions are the 2-way and 4-way UCs, of which the 4-way construction results in an asymptotically smaller size.

So far only Valiant’s 2-way UC has been implemented and the not yet implemented 4-way UC was postulated to be the most efficient one.

## 1.3 Outline and Our Contributions

In summary, we provide the first implementation and detailed evaluation of Valiant’s 4-way UC and propose an even more efficient hybrid UC. We elaborate on the size of the generalized  $k$ -way UCs for  $k \neq 2$  and  $k \neq 4$ .

After revisiting Valiant’s UC construction [Val76, KS16] and its  $k$ -way generalization [LMS16] in Sect. 2, we provide the following contributions:

**Our modular programming algorithm (Sect. 3):** We detail a modular algorithm that provides the description of the input function  $f$  as program bits  $p_f$  to the UC, both for Valiant’s 4-way UC as well as for the  $k$ -way UC of Lipmaa et al. [LMS16].

**New universal circuit constructions (Sect. 4):** We start with a new 3-way UC. After providing modular building blocks for this UC, we show that it is asymptotically larger than Valiant’s UCs. Then, we propose a *hybrid UC construction* that can efficiently combine  $k$ -way constructions for multiple values of  $k$ .<sup>2</sup> With this, we combine Valiant’s 2-way and 4-way UCs to achieve the smallest UC known so far.

**Size of UCs (Sect. 5):** We compare the asymptotic and concrete sizes of Valiant’s (2-way and 4-way) UCs and that of different  $k$ -way UCs. We show that of all  $k$ -way UCs, Valiant’s 4-way UC provides the best results for large circuits. Moreover, our hybrid UC in most cases improves over the 2-way UC by up to around 4.5% in its size, and over the 4-way UC by up to 2% (for large input circuits). In Table 1 we compare the concrete communication of PFE using SFE and our new UC implementation to the previous works on special-purpose OT-based PFE protocols.

**Implementation of Valiant’s 4-way UC and experiments (Sect. 6):** We implement Valiant’s 4-way UC and describe how our implementation can directly be used in the PFE framework of [KS16]. We experimentally evaluate the performance of our UC generation and programming algorithm with a set of example circuits and compare it on the same platform with the 2-way UC compiler of [KS16].

**Table 1.** Comparison of overall communication between special-purpose PFE protocols and UC-based ones for simulated circuits of size  $n$ . The numbers are for 128 bit symmetric security. The underlying SFE protocol for UC-based PFE is Yao’s protocol [Yao86] with the garbled row reduction optimization [NPS99] and X- and Y-switching blocks are instantiated using free XORs as described in [KS08a]. This yields one ciphertext per X- and Y-switching block, and three ciphertexts per universal gate.

$n$	Special-purpose PFE		UC-based PFE using Yao		
	[MS13]	[BBKL17]	2-way UC [KS16]	Our 4-way UC	Our hybrid UC
$10^3$	3.5 MB	2.0 MB	0.6 MB	0.6 MB	0.6 MB
$10^4$	44.8 MB	26.3 MB	8.4 MB	8.4 MB	8.2 MB
$10^5$	549.6 MB	324.0 MB	109.6 MB	107.8 MB	106.2 MB
$10^6$	6 509.9 MB	3 847.9 MB	1 360.3 MB	1 308.4 MB	1 308.4 MB
$10^7$	75 236.5 MB	44 562.1 MB	16 038.8 MB	15 677.7 MB	15 413.7 MB

<sup>2</sup> Our hybrid UC is orthogonal to that of [KS16], who combine Valiant’s UC with building blocks from [KS08b] for the inputs and outputs.

## 2 Preliminaries

In this section, we summarize the existing UC constructions. We provide necessary background information in Sect. 2.1, explain Valiant's construction [Val76] in Sect. 2.2 and the improvements of [KS16, LMS16] on the 2-way, 4-way and  $k$ -way UCs in Sects. 2.3, 2.4 and 2.5, respectively.

### 2.1 Preliminaries to Valiant's UC Constructions

Let  $G = (V, E)$  be a *directed graph* with set of *nodes*  $V$  and *edges*  $E \subseteq V \times V$ . The number of incoming [outgoing] edges of a node is called its *indegree* [*outdegree*]. A graph has *fanin* [*fanout*]  $d$  if the indegree [outdegree] of all its nodes is at most  $d$ . In the following, we denote by  $\Gamma_d(n)$  the set of all acyclic graphs with fanin and fanout  $d$  having  $n$  nodes. Similarly, the fanin [fanout] of a circuit can be defined based on the maximal number of incoming [outgoing] wires of all its gates, inputs and outputs.

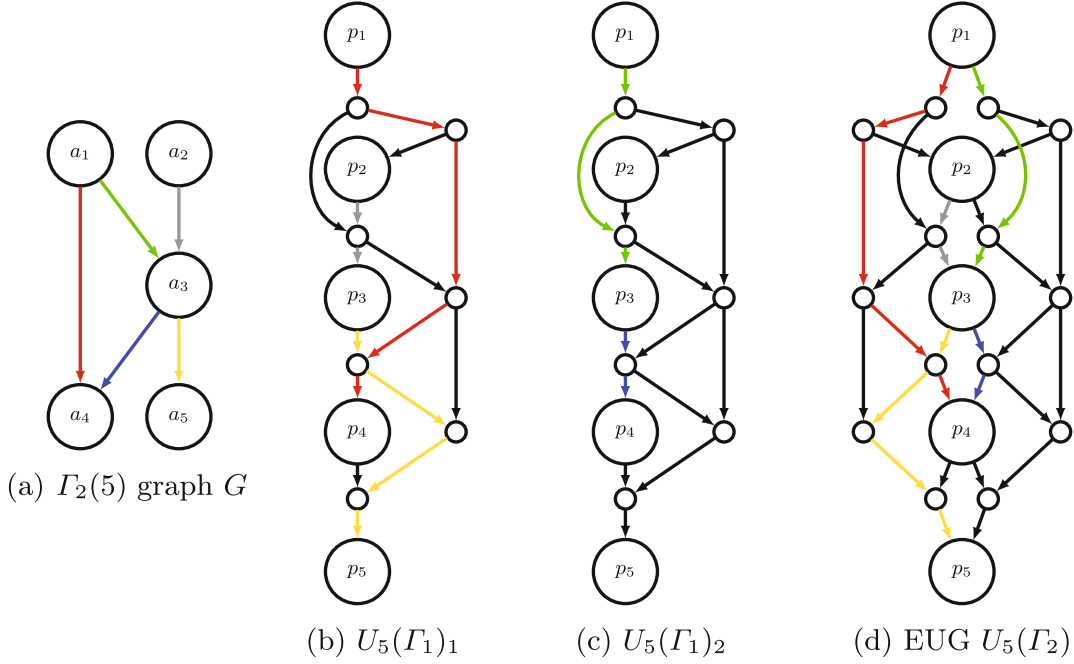
Let  $G = (V, E) \in \Gamma_d(n)$ . A mapping  $\eta^G : V \rightarrow \{1, \dots, n\}$  is called *topological order* if  $(a_i, a_j) \in E \Rightarrow \eta^G(a_i) < \eta^G(a_j)$  and  $\forall a_1, a_2 \in V : \eta^G(a_1) = \eta^G(a_2) \Rightarrow a_1 = a_2$ . A topological order in  $G \in \Gamma_d(n)$  can be found with computational complexity  $\mathcal{O}(dn)$ .

A circuit  $C_{u,v}^{k*}$  with  $u$  inputs,  $k^*$  gates and  $v$  outputs and fanin or fanout  $d > 2$  can be reduced to a circuit with fanin and fanout 2. Shannon's expansion theorem [Sha49, Sch08] describes how gates with larger fanin can be reduced to gates with two inputs by adding additional gates. [Val76, KS16] describe adding copy gates in order to eliminate larger fanout and elaborate on the implied overhead ( $k \leq 2k^* + v$ ). [KS08b, KS16] implement these methods and we thus assume that our input Boolean circuit  $C_{u,v}^k$  has fanin and fanout 2 for all its  $u$  inputs,  $k$  gates and  $v$  outputs. We transform  $C_{u,v}^k$  into a  $\Gamma_2(n)$  graph  $G$  with  $n = u + v + k$  by creating a node for each input, gate and output, and an edge for each wire in  $C_{u,v}^k$ .

*Edge-embedding* is a mapping from graph  $G = (V, E)$  into  $G' = (V', E')$  with  $V \subseteq V'$  and  $E'$  containing a path for each  $e \in E$ , such that the paths are pairwise edge-disjoint. A graph  $U_n(\Gamma_d) = (V_U, E_U)$  is an *Edge-Universal Graph* (EUG) for  $\Gamma_d(n)$  if every graph  $G \in \Gamma_d(n)$  can be edge-embedded into  $U_n(\Gamma_d)$ .<sup>3</sup>  $U_n(\Gamma_d)$  has distinguished nodes called *poles*  $\{p_1, \dots, p_n\} \subseteq V_U$  where each node  $a \in V$  is mapped to exactly one pole with a mapping  $\varphi$ , such that every node in  $G$  has a corresponding pole in  $U_n(\Gamma_d)$ . This mapping is defined by a concrete topological order  $\eta^G$  of the original graph  $G$ , i.e.,  $\varphi : V \rightarrow V_U$  with  $\varphi(a) = p_{\eta^G(a)}$ . Besides the poles,  $U_n(\Gamma_d)$  might have additional nodes that enable the edge-embedding. For each edge  $(a_i, a_j) \in E$  we then define a disjoint path between the corresponding poles  $(\varphi(a_i), \dots, \varphi(a_j)) = (p_{\eta^G(a_i)}, \dots, p_{\eta^G(a_j)})$  in  $U_n(\Gamma_d)$ , i.e., without using any edge in  $U_n(\Gamma_d)$  in more than one path.

Let  $U_n(\Gamma_1)$  be an EUG for graphs in  $\Gamma_1(n)$  with poles  $P = \{p_1, \dots, p_n\}$ . The poles have fanin and fanout 1, while all other nodes have fanin and fanout 2. An

<sup>3</sup> For the sake of simplicity, we denote this graph with  $U_n(\Gamma_d)$  instead of  $U(\Gamma_d(n))$ .



**Fig. 1.** (a) shows an example  $I_2(5)$  graph  $G$ . (b)–(c) show the edge-embedding of  $G$  into two  $U_5(I_1)$  instances with poles  $(p_1, \dots, p_5)$ . (d) shows the edge-embedding of  $G$  into one  $U_5(I_2)$  graph.

EUG  $U_n(I_d)$  for  $d \geq 2$  can be created by taking  $d$  instances of  $U_n(I_1)$  EUGs, and merging each pole  $p_i$  with its multiple instances, allowing the poles to have fanin-fanout  $d$ . Let  $U_n(I_d) = (V'_U, E'_U)$  be an EUG with fanin and fanout  $d$ , with  $U_n(I_1)_1 = (V_1, E_1), \dots, U_n(I_1)_d = (V_d, E_d)$ .  $P$  contains the merged poles and  $V'_U = P \cup_{i=1}^d V_i \setminus P_i$  and  $E'_U = \cup_{i=1}^d E_i$ .

We give an example for better understanding. Let  $G = (V, E)$  be the graph with 5 nodes in Fig. 1a. Our aim is to edge-embed  $G$  into EUG  $U_5(I_2)$ . Therefore, we use two instances of  $U_5(I_1)$ :  $U_5(I_1)_1$  in Fig. 1b and  $U_5(I_1)_2$  in Fig. 1c. The edges  $(a_1, a_4)$ ,  $(a_2, a_3)$  and  $(a_3, a_5)$  are embedded in  $U_5(I_1)_1$ , and the edges  $(a_1, a_3)$  and  $(a_3, a_4)$  in  $U_5(I_1)_2$ . Merging the poles of  $U_5(I_1)_1$  and  $U_5(I_1)_2$  produces  $U_5(I_2)$  shown in Fig. 1d.

## 2.2 Valiant's UC Constructions

The size of a function  $f$  represented by a circuit  $C_{u,v}^k$  with fanin and fanout 2 is  $n = u + v + k$ . In the following, we describe Valiant's UC construction [Val76, Weg87] that can be programmed to evaluate any function of size  $n$ . Circuit  $C_{u,v}^k$  is represented as a graph  $G \in I_2(n)$  (cf. Sect. 2.1).

Valiant's UC is based on an EUG  $U_n(I_2) = (V_U, E_U)$  with fanin and fanout 2, which can be transformed to a Boolean circuit.  $P \subseteq V_U$  contains the poles of  $U_n(I_2)$  (cf. Sect. 2.1). Poles  $\{1, \dots, u\}$  correspond to the inputs,  $\{(u+1), \dots, (u+k)\}$  to the gates,  $\{(u+k+1), \dots, n\}$  to the outputs of  $C_{u,v}^k$ . The edges of the graph of the circuit  $G = (V, E)$  have to be embedded into  $U_n(I_2)$ . After the transformations described in Sect. 2.1, every node in  $G$  has fanin and



fanout 2, and we denote a topological order on  $V$  by  $\eta^G$ . We briefly describe the edge-embedding process in Sects. 2.3 and 3.

**Translating a  $U_n(\Gamma_2)$  into a Universal Circuit.** Every node  $w \in V_U$  fulfills a task when  $U_n(\Gamma_2)$  is translated to a UC. Programming the UC means specifying its control bits along the paths defined by the edge-embedding and by the gates of circuit  $C_{u,v}^k$ . Depending on the number of incoming and outgoing edges and its type, a node is translated to:

- G1.** If  $w$  is a pole and corresponds to an input or output in  $G$ , then  $w$  is an *input or output* in  $U_n(\Gamma_2)$  as well.
- G2.** If  $w$  is a pole and corresponds to a gate in  $G$ ,  $w$  is programmed as a *universal gate* (UG). A 2-input UG supports any of the 16 possible gate types represented by the 4 control bits of the gate table  $(c_1, c_2, c_3, c_4)$ . It implements function  $ug: \{0, 1\}^2 \times \{0, 1\}^4 \rightarrow \{0, 1\}$  that computes:

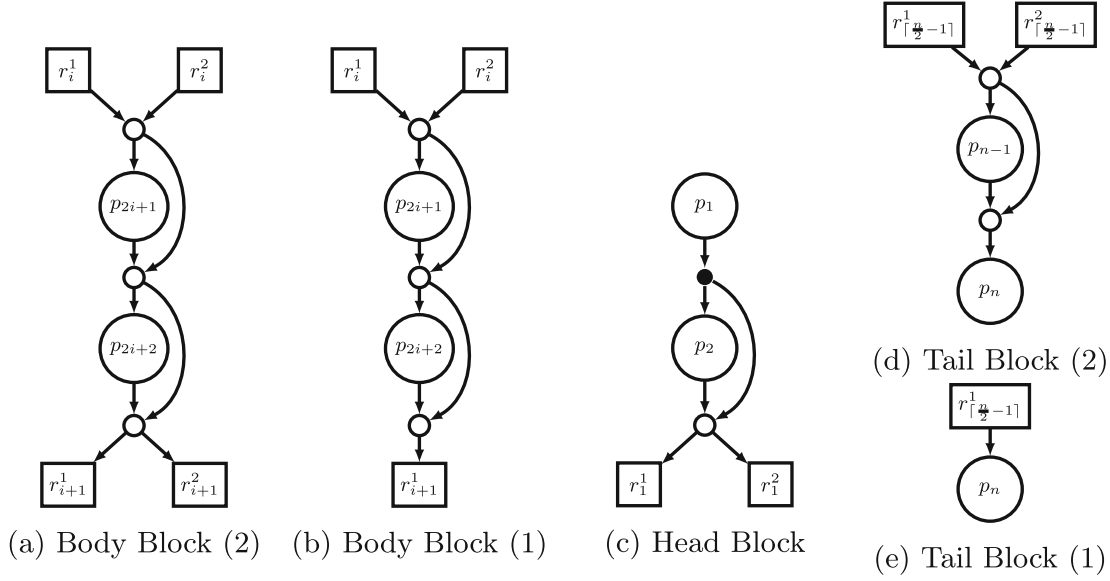
$$ug(x_1, x_2, c_1, c_2, c_3, c_4) = \overline{x_1}\overline{x_2}c_1 + \overline{x_1}x_2c_2 + x_1\overline{x_2}c_3 + x_1x_2c_4. \quad (1)$$

Generally, a UG can be implemented with 3 AND and 6 XOR gates (resp. with a two-input gate when using Yao's protocol for SFE) [KS16].

- G3.** If  $w$  is no pole and has indegree and outdegree 2,  $w$  is programmed as an *X-switching block*, that computes  $f_X : \{0, 1\}^2 \times \{0, 1\} \rightarrow \{0, 1\}^2$  with  $f_X((x_1, x_2), c) = (x_{1+c}, x_{2-c})$ . This block can be implemented with 1 AND and 3 XORs (resp. a one-input gate with Yao) [KS08a].
- G4.** If  $w$  is no pole and has indegree 2 and outdegree 1,  $w$  is programmed as a *Y-switching block* that computes  $f_Y : \{0, 1\}^2 \times \{0, 1\} \rightarrow \{0, 1\}$  with  $f_Y((x_1, x_2), c) = x_{1+c}$ . This block can be implemented with 1 AND and 2 XORs (resp. a one-input gate with Yao) [KS08a].
- G5.** If  $w$  is no pole and has indegree 1 and outdegree 2, it has been placed to copy its input to its two outputs. Therefore, when translated to a UC,  $w$  is replaced by multiple outgoing wires in the parent node [KS16], since the UC itself does not have the fanout 2 restriction. In  $U_n(\Gamma_2)$ ,  $w$  is added due to the fanout 2 restriction in the EUG.
- G6.** If  $w$  is no pole and has indegree and outdegree 1,  $w$  is removed and replaced by a wire between its parent and child nodes.

The nodes programmed as UG (G2), X-switching block (G3) or Y-switching block (G4) are so-called programmable blocks. This means that a programming bit or vector is necessary besides the two inputs to define their behavior as described above. These programming bits and vectors that build up the programming of the UC  $p_f$  are defined by the paths in the edge-embedding of  $G$  (the graph of circuit  $C_{u,v}^k$  describing  $f$ ) into  $U_n(\Gamma_2)$ .

**Recursion Base.** Valiant's construction is recursive, and the recursion base is reached when the number of poles is between 1 and 6. These recursion base graphs are shown in [Val76, KS16].  $U_1(\Gamma_1)$  is a single pole,  $U_2(\Gamma_1)$  and  $U_3(\Gamma_1)$  are two and three connected poles, respectively.  $U_4(\Gamma_1)$ ,  $U_5(\Gamma_1)$  and  $U_6(\Gamma_1)$  are constructed with 3, 7 and 9 additional nodes, respectively.



**Fig. 2.** (a) shows Valiant's 2-way EUG  $U_n^{(2)}(\Gamma_1)$  [Val76]. (c) shows the corresponding head block, (b) and (d)–(e) show body and tail blocks, respectively, for different numbers of poles.

### 2.3 Valiant's 2-Way UC Construction

We described in Sect. 2.1 that a  $U_n(\Gamma_d)$  EUG can be constructed of  $d$  instances of  $U_n(\Gamma_1)$  EUGs. Therefore, Valiant provides an EUG for  $\Gamma_1(n)$  graphs, two of which can build an EUG for  $\Gamma_2(n)$  graphs. Let  $P = \{p_1, \dots, p_n\}$  be the set of poles in  $U_n^{(2)}(\Gamma_1)$  that have indegree and outdegree 1. Valiant's 2-way EUG construction for  $\Gamma_1(n)$  graphs of size  $\sim 2.5n \log_2 n$  is shown in Fig. 2, where we emphasize the poles as large circles and the additional nodes as small circles or rectangles. The corresponding UC has twice the size  $\sim 5n \log_2 n$ , since it corresponds to the EUG for  $\Gamma_2(n)$  graphs.

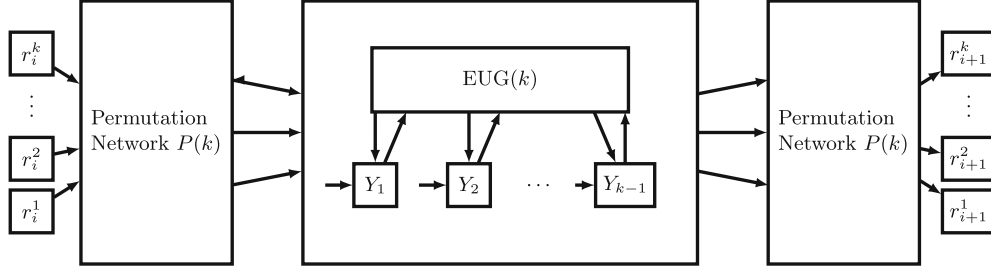
The rectangles are special nodes that build the set of poles in the next recursion step, i.e.,  $R_{\lceil \frac{n}{2}-1 \rceil}^1 = \{r_1^1, \dots, r_{\lceil \frac{n}{2}-1 \rceil}^1\}$ ,  $R_{\lceil \frac{n}{2}-1 \rceil}^2 = \{r_1^2, \dots, r_{\lceil \frac{n}{2}-1 \rceil}^2\}$ . Another EUG is built with these poles which produces new subgraphs with size  $\lceil \frac{\lceil \frac{n}{2}-1 \rceil}{2} - 1 \rceil$ , s.t. we have four subgraphs at this level.

This construction is called the *2-way EUG or UC construction*. An open-source implementation of this construction optimized for PFE is provided in [KS16]. Independently, [LMS16] also implemented this 2-way UC, additionally optimizing for the total number of gates.

### 2.4 Valiant's 4-Way UC Construction

Valiant provides another, so-called *4-way EUG or UC construction* [Val76].  $U_n^{(4)}(\Gamma_1)$  has a 4-way recursive structure, i.e., nodes in special sets  $R_{\lceil \frac{n}{4}-1 \rceil}^1$ ,  $R_{\lceil \frac{n}{4}-1 \rceil}^2$ ,  $R_{\lceil \frac{n}{4}-1 \rceil}^3$  and  $R_{\lceil \frac{n}{4}-1 \rceil}^4$  are the poles in the next recursion step (cf. Fig. 4a). The recursion base is the same as in Sect. 2.2. This construction results in UCs of





**Fig. 3.**  $k$ -way EUG construction  $U_n^{(k)}(\Gamma_1)$  [LMS16].

smaller size  $\sim 4.75n \log_2 n$  but has not been implemented before due to its more complicated structure.

## 2.5 Lipmaa et al.’s Generalized $k$ -Way UC Construction

In [LMS16], Lipmaa et al. generalize Valiant’s approach by providing a UC with any number of recursion points  $k$ , the so-called  $k$ -way EUG or UC construction. We note that their construction slightly differs from Valiant’s EUG construction, since they do not consider the restriction on the fanout of the poles, i.e., the nodes in the EUG that correspond to universal gates or inputs (cf. Sect. 2.2). This optimization has also been included in [KS16] when translating an EUG to a UC, but including it in the block design leads to better sizes for the number of XOR gates.

The idea is to split  $n = u + v + k$  in  $m = \lceil \frac{n}{k} \rceil$  blocks as shown in Fig. 3. Every block  $i$  consists of  $k$  inputs  $r_i^1, r_i^2, \dots, r_i^k$  and  $k$  outputs  $r_{i+1}^1, r_{i+1}^2, \dots, r_{i+1}^k$  as well as  $k$  poles, except for the last block which has a number of poles depending on  $n \bmod k$ . For every  $j \leq k$ , the list of all  $r_i^j$  builds the poles of the  $j^{\text{th}}$  subgraph of the next recursion step, i.e. we have  $k$  subgraphs. Additionally, every block begins and ends with a Waksman permutation network [Wak68] such that the inputs and outputs can be permuted to every pole. A Y-switching block is placed in front of every pole  $p_i$  which is connected to the  $i^{\text{th}}$  output of the permutation network as well as the  $i^{\text{th}}$  output of a block-internal EUG  $U_k(\Gamma_1)$ . Thus, [LMS16] reduce the problem of finding an efficient  $k$ -way EUG  $U_n^{(k)}(\Gamma_2)$  to the problem of finding the smallest EUG  $U_k(\Gamma_1)$ . Their solution is to build the block-internal EUG with the UC construction of [KS08b], which was claimed to be more efficient for smaller circuits than [Val76]. However, they calculate the optimal  $k$  value to be around 3.147, which implies that the best solutions are found using small EUGs, for which Valiant provides hand-optimized solutions (i.e., for  $k = 2, 3, 4, 5, 6$ ) [Val76].

## 3 Our Modular Edge-Embedding Algorithm

The detailed embedding algorithm and the open-source UC implementation of [KS16] was specifically built for the 2-way UC, dealing with the whole UC skeleton as one block. In contrast, based on the modular design of [LMS16], we modularize the edge-embedding task into multiple sub-tasks and describe how they

can be performed separately. In this section, we detail this modular approach for edge-embedding a graph into Valiant's 4-way EUG: the edge-embedding can be split into two parts, which are then combined. In Sect. 3.1, we describe our modular approach based on the edge-embedding algorithm of [KS16] for Valiant's 2-way EUG. This can be generalized to any  $2^i$ -way EUG construction. Moreover, the same algorithm can be applied with a few modifications for Lipmaa et al.'s  $k$ -way recursive generalization [LMS16], which we describe in Sect. 3.2.

### 3.1 Edge-Embedding in Valiant's 4-Way UC

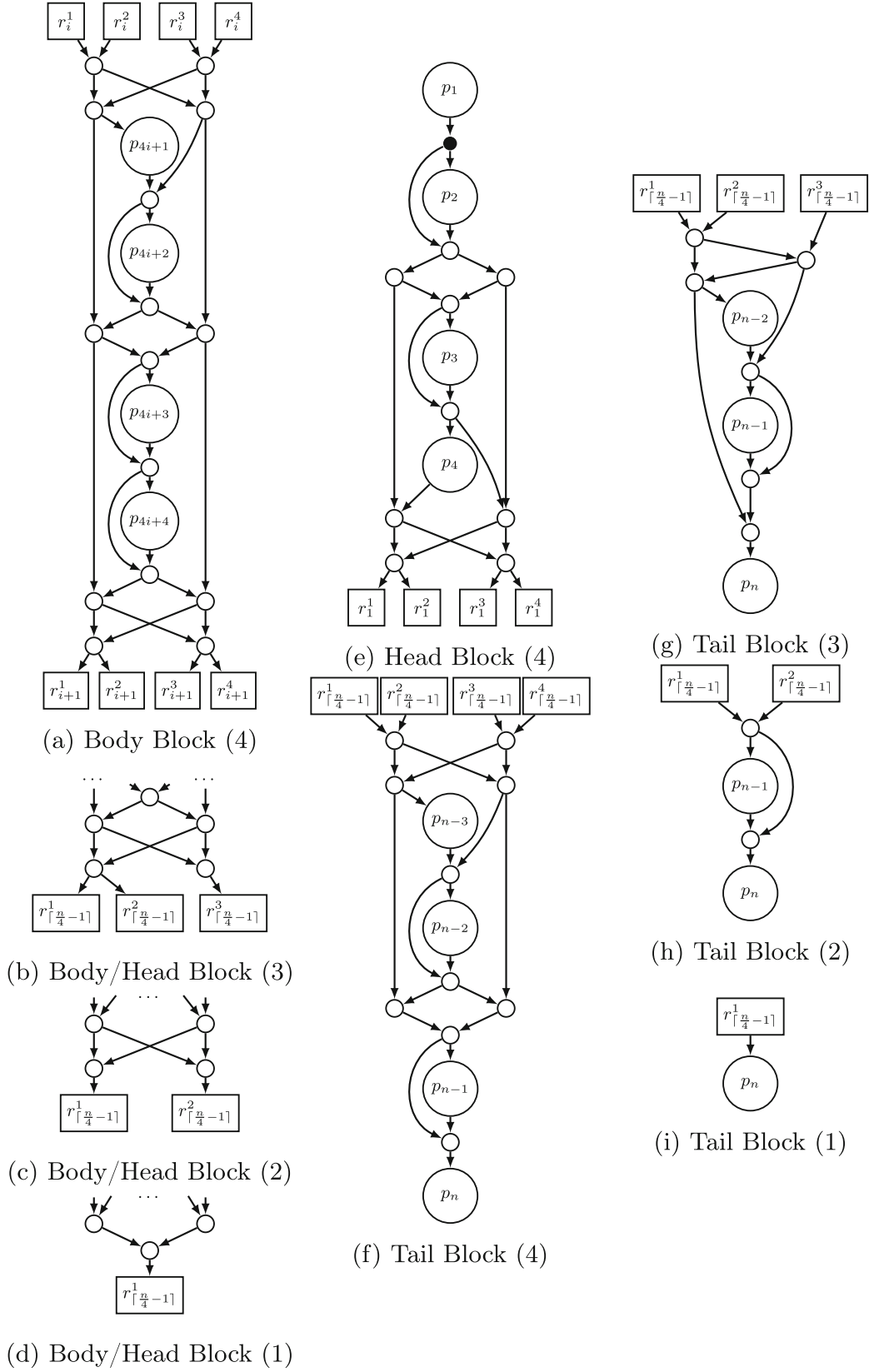
Similar to the 2-way EUG construction (cf. Sect. 2.3), Valiant provides a 4-way EUG construction for  $\Gamma_1(n)$  graphs which can be extended to an EUG for  $\Gamma_2(n)$  graphs by utilizing two instances  $U_n^{(4)}(\Gamma_1)_1$  and  $U_n^{(4)}(\Gamma_1)_2$  as described in Sect. 2.1. The construction with our optimizations is visualized in Fig. 4. Valiant offers the main, so-called *Body Block* (Fig. 4a) consisting of 4 poles (large circles), 15 nodes (small circles) as well as 8 recursion points (squares). These body blocks are connected such that the 4 top [bottom] recursion points of one block are the 4 bottom [top] recursion points of the next block. Similarly to the 2-way EUG, 4 sets are created for  $n$  nodes, i.e.,  $R_{\lceil \frac{n}{4} - 1 \rceil}^1 = \{r_1^1, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^1\}$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^2 = \{r_1^2, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^2\}$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^3 = \{r_1^3, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^3\}$ , and  $R_{\lceil \frac{n}{4} - 1 \rceil}^4 = \{r_1^4, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^4\}$  which are the poles of 4  $U_{\lceil \frac{n}{2} \rceil - 1}(\Gamma_1)$  EUGs in the next recursion step. Then, these also create 4 subgraphs until the recursion base is reached, cf. Sect. 2.2.

We note that the top [bottom] block does not need the upper [lower] recursion points since its poles are the inputs [outputs] in the block. Therefore, we provide so-called Head and Tail Blocks. A *Head Block* occurs at the top of a chain of blocks (cf. Fig. 4e), it has 4 poles, no inputs, 4 output recursion points and 10 nodes, of which the first one (denoted by a filled circle) has one input and therefore translates to wires in the UC.

As a counterpart, *Tail Blocks* occur at the bottom of a chain of blocks, have at most 4 poles, 4 input recursion points, no outputs and at most 10 nodes depending on the number of poles. The 4 tail block constructions are depicted in Figs. 4f–i and are used, based on the remainder of  $n$  modulo 4, with the respective body or head blocks when  $n \in \{5, 6, 7\}$ , the lower parts of which are shown in Figs. 4a–d.

**Block Edge-Embedding.** In this first part of the edge-embedding process, we consider the 4 top [bottom] recursion points of the block as intermediate nodes where the inputs [outputs] of the block enter [leave]. The blocks are built s.t. any of these inputs can be forwarded to exactly one of the 4 poles of the block and the output of any pole can be forwarded to exactly one output or another pole having a higher topological order.

We formalize this behaviour as follows: In  $U_n^{(4)}(\Gamma_1) = (V_U, E_U)$ , let  $B$  be the block visualized in Fig. 4a with poles  $p_{4i+1}, \dots, p_{4i+4}$ . Let mapping  $\eta^U : V_U \rightarrow \mathbb{N}^+$  denote a topological order of  $V_U$ . Then, the nodes  $r_i^1, \dots, r_i^4$  and  $r_{i+1}^1, \dots, r_{i+1}^4$  denote the input and output recursion points of block  $B$ . Additionally, let  $in =$



**Fig. 4.** (a) shows Valiant's 4-way EUG  $U_n^{(4)}(\Gamma_1)$  [Val76]. (e) shows our head block construction, (a)–(d) and (f)–(i) show our body and tail block constructions, respectively, for different numbers of poles.

$(in_1, \dots, in_4) \in \{0, \dots, 4\}^4$  and  $out = (out_1, \dots, out_4) \in \{0, \dots, 7\}^4$  denote the input and output vectors of  $B$ . The value 0 of the input and output vectors is a *dummy value* which is used if an input [a pole] is not forwarded to any pole [output] of  $B$ . The output vector has a larger value range, since a pole can be forwarded to another pole or an output recursion point. Therefore, we use values 1, 2 and 3 for poles  $p_2, p_3$  and  $p_4$  and values 4, 5, 6 and 7 for the output recursion points. Pole  $p_1$  cannot be a destination for a path in  $B$ , since  $\eta^U(p_1)$  is less than the topological order of any other pole in  $B$ . Additionally, the values of  $in$  and  $out$  need to be pairwise different or 0. Every combination of input and output vector covering the conditions formalized below in Eqs. 2–6 are valid for  $B$ . A pair  $(r_i^l, p_j) \in \mathcal{P}$  or  $(p_j, r_{i+1}^l) \in \mathcal{P}$  is a path from  $r_i^l$  to  $p_j$  or  $p_j$  to  $r_{i+1}^l$  in the set of all paths  $\mathcal{P}$  in  $B$ . Then,  $\mathcal{P}_B \subseteq \mathcal{P}$  denote the paths that are to be edge-embedded (cf. Sect. 6.1).

$$\forall l \in \{1, \dots, 4\} : in_l \neq 0 \rightarrow (r_i^l, p_{in_l}) \in \mathcal{P}_B, \quad (2)$$

$$out_l \neq 0 \wedge out_l < 4 \rightarrow (p_j, p_{1+out_l}) \in \mathcal{P}_B \wedge \eta^U(p_j) < \eta^U(p_{1+out_l}) \quad (3)$$

$$out_l > 3 \rightarrow (p_j, r_{i+1}^{l-3}) \in \mathcal{P}_B. \quad (4)$$

$$\forall in_i, in_j \in in : i \neq j \rightarrow in_i = 0 \vee in_i \neq in_j. \quad (5)$$

$$\forall out_i, out_j \in out : i \neq j \rightarrow out_i = 0 \vee out_i \neq out_j. \quad (6)$$

**Recursion Point Edge-Embedding.** The block edge-embedding covers only the programming of the nodes within a block. Another task left is to program the recursion points. We use the supergraph construction of [KS16] which, in every step, splits a  $\Gamma_2(n)$  graph in two  $\Gamma_1(n)$  graphs, which are merged to two  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graphs. [KS16] use this for defining the paths in Valiant’s 2-way EUG. For Valiant’s 4-way EUG, we use every second step of their algorithm with a minor modification.

Let  $C_{u,v}^k$  be the Boolean circuit computing function  $f$  that our UC needs to compute, and  $G \in \Gamma_2(n)$  its graph representation (cf. Sect. 2.2).

1. *Splitting  $G \in \Gamma_2(n)$  in two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ :* As described in Sect. 2.1, Valiant’s UC is derived from an EUG for  $\Gamma_2(n)$  graphs, which consists of two EUGs for  $\Gamma_1(n)$  graphs merged by their poles. Therefore,  $G$  is split into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ .  $G_1$  and  $G_2$  then need to be edge-embedded into EUGs  $(U_n^{(4)}(\Gamma_1))_1$  and  $(U_n^{(4)}(\Gamma_1))_2$ , respectively.  $G = (V, E) \in \Gamma_2(n)$  is split by 2-coloring its edges as described in [Val76, KS16], which can always be done due to König’s theorem [Kö31, LP09]. After 2-coloring,  $E$  is divided to sets  $E_1$  and  $E_2$ , using which we build  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$ , with the following conditions:

$$\forall e \in E : (e \in E_1 \vee e \in E_2) \wedge \neg(e \in E_1 \wedge e \in E_2). \quad (7)$$

$$\forall e = (v_1, v_2) \in E_1 : \neg \exists e' = (v_3, v_4) \in E_1 : v_2 = v_4 \vee v_1 = v_3. \quad (8)$$

$$\forall e = (v_1, v_2) \in E_2 : \neg \exists e' = (v_3, v_4) \in E_2 : v_2 = v_4 \vee v_1 = v_3. \quad (9)$$

2. *Merging a  $\Gamma_1(n)$  graph into a  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graph:* In an EUG, the number of poles decreases in each recursion step and therefore, merging a  $\Gamma_1(n)$

graph into a  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graph provides information about the paths to be taken. Let  $G_1 = (V, E) \in \Gamma_1(n)$  be a topologically ordered graph and  $G_m = (V', E') \in \Gamma_2(\lceil \frac{n}{2} \rceil)$  be a graph with nodes  $v'_1, \dots, v'_{\lceil \frac{n}{2} \rceil}$ . We define two labellings  $\eta_{\text{in}}$  and  $\eta_{\text{out}}$  on  $G_m$  with  $\eta_{\text{in}}(v_i) = i$  and  $\eta_{\text{out}}(v_i) = \eta_{\text{in}}(v_i) - 1 = i - 1$ . Additionally, we define a mapping  $\theta_V$  that maps a node  $v_i \in V$  to a node  $v_j \in V'$  with  $\theta_V(v_i) = v'_{\lceil \frac{i}{2} \rceil}$ . That means two nodes in  $G_1$  are mapped to one node in  $G_m$ . At last, we define a mapping  $\theta_E$  that maps an edge  $e_i = (v_i, v_j) \in E$  to an edge  $e_j \in E'$  with  $\theta_E((v_i, v_j)) = (v_{\eta_{\text{in}}(\theta_V(v_i))}, v_{\eta_{\text{out}}(\theta_V(v_j))})$ . That means every edge in  $G_1$  is mapped to an edge in  $G_m$  as follows:  $e = (v_i, v_j) \in E$  is mapped to  $e' = (v'_k, v'_l) \in E'$ , s.t.  $v'_k = \theta_V(v_i)$ , but  $v'_l$  is not the new node of  $v_j$  in  $G_m$  but  $v'_{l+1}$ .  $G_m$  is built as follows:  $V' = \{v'_1, \dots, v'_{\lceil \frac{n}{2} \rceil}\}$  and  $E' = \bigcup_{e \in E} \theta_E(e)$ . Then for all  $e = (v'_i, v'_j) \in E'$  and  $j < i$ ,  $e$  is removed from  $E'$ , along with the last node  $v'_{\lceil \frac{n}{2} \rceil}$  (due to the definition of  $\theta_E$ , it does not have any incoming edges). The resulting  $G_m$  is a topologically ordered graph in  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$ .

3. *The supergraph for Valiant's 4-way EUG construction:* In the first step,  $G$  is split to two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ .  $G_1$  and  $G_2$  contain all the edges that should be embedded as paths between poles in the first and second EUGs for  $\Gamma_1(n)$ , respectively. We now explain how to edge-embed the  $\Gamma_1(n)$  graph  $G_1$  into an EUG  $U_n^{(4)}(\Gamma_1)$  (for  $G_2$  it is the same).

For embedding in a 2-way UC,  $G_1$  is firstly merged to a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph  $G_m$ .  $G_m$  is then 2-colored and split into two  $\Gamma_1(\lceil \frac{n}{2} \rceil)$  graphs  $G_1^1$  and  $G_1^2$  [KS16]. These get merged to two  $\Gamma_2(\lceil \frac{\frac{n}{2}-1}{2} - 1 \rceil)$  graphs  $G_m^1$  and  $G_m^2$ .  $G_1^1$  is the first and  $G_1^2$  is the second subgraph of  $G_1$ . Then  $G_1^{\psi \circ 1}$  and  $G_1^{\psi \circ 2}$  denote the first and second subgraph of  $G_1^\psi$ , respectively. These steps are repeated until the  $\Gamma_1$  subgraphs have at most 4 nodes.

In Valiant's 4-way EUG construction [Val76], a supergraph that creates 4 subgraphs in each step is necessary. We require a merging method where a  $\Gamma_1(n)$  graph is merged to a  $\Gamma_4(\lceil \frac{n}{4} - 1 \rceil)$  graph where 4 nodes build a new node, and 4-color this graph to retrieve 4 subgraphs. However, this can directly be solved by using the method described above from [KS16]: after repeating the 2-coloring and the merging twice, we gain 4 subgraphs ( $G_1^{11}$ ,  $G_1^{12}$ ,  $G_1^{21}$  and  $G_1^{22}$ ). These can be used as if they were the result of 4-coloring the graph obtained by merging every 4 nodes into one.

However, there is a modification in this case: the first 2-coloring is a preprocessing step, which does not map to an EUG recursion step. Therefore, we have to define another labelling  $\eta_{\text{out}_P}(v) = \eta_{\text{in}}(v)$ , since in this preprocessing step we need to keep node  $v_{\lceil \frac{n}{2} \rceil}$ . Then the creation of the supergraph for the 4-way EUG construction works as follows: We merge  $G_1$  to a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph with labelling  $\eta_{\text{in}}$  and  $\eta_{\text{out}_P}$  and get  $G_m$ . After that, we split  $G_m$  into two  $\Gamma_1(\lceil \frac{n}{2} \rceil)$  graphs  $G_1^1$  and  $G_1^2$ . These get merged to  $\Gamma_2(\lceil \frac{n}{4} \rceil - 1)$  graphs  $G_m^1$  and  $G_m^2$  using the  $\eta_{\text{in}}$  and  $\eta_{\text{out}}$  labellings. Finally, these two graphs get splitted into 4  $\Gamma_1(\lceil \frac{n}{4} - 1 \rceil)$  graphs  $G_1^{11}$ ,  $G_1^{12}$ ,  $G_1^{21}$  and  $G_1^{22}$ . These are the relevant graphs for the first recursion

**Listing 1.** Edge-embedding algorithm for Valiant's 4-way EUG

```

1  procedure edge-embedding ( $U, G_1 = (V, E)$ )
2  Let  $S$  be the set of the 4  $\Gamma_1$  subgraphs of  $G_1$  in the supergraph
3  Let  $R$  be the 4 recursion step graphs
4  Let  $B$  be the set of blocks in  $U$ 
5  for all  $e = (v_i, v_j) \in E$  do
6    Let  $i'$  and  $j'$  denote the positions of  $v_i$  and  $v_j$  in their blocks
7     $b_i \leftarrow \lceil \frac{i}{4} \rceil, b_j \leftarrow \lceil \frac{j}{4} \rceil$  // number of block in which  $v_i$  and  $v_j$  are
8    Let  $out$  [ $r_1$ ] denote the output vector [recursion points] of  $B_{b_i}$ 
9    Let  $in$  [ $r_0$ ] denote the the input vector [recursion points] of  $B_{b_j}$ 
10   if  $b_i = b_j$  do //  $v_i$  and  $v_j$  are in the same block
11     if  $v_i \neq v_j$  do
12        $out_{i'} \leftarrow j' - 1$ 
13     end if
14   else //  $v_i$  and  $v_j$  are in different blocks
15     Let  $s = (V', E') \in S$  denote the  $\Gamma_1$  graph with  $e' = (p_{b_i}, p_{b_j-1}) \in E'$  and
16        $\hookrightarrow e'$  is not marked
17     Mark  $e'$ 
18     Let  $x$  denote the number with  $s = S_x$ 
19     Set the control bit of  $r_0^x$  to 1
20     if  $b_j = b_i + 1$  do //  $b_j$  and  $b_i$  are neighbours
21        $y \leftarrow 0$ 
22     else
23        $y \leftarrow 1$ 
24     end if
25     Set the control bit of  $r_1^x$  to  $y$ 
26      $out_{i'} \leftarrow x + 4, in_x \leftarrow j'$ 
27   end if
28 end for
29 Edge-embed all blocks in  $U$  // edge-embed all sub-blocks
30 for  $i = 1$  to 4 do
31   if  $S_i$  exists do
32     call edge-embedding( $R_i, S_i$ )
33   end if
34 end for
end procedure

```

step in Valiant's 4-way EUG construction. Now we continue for all 4 subgraphs until we reach the recursion base with 4 or less nodes.

**4-Way Edge-Embedding Algorithm.** In Listing 1, we combine block edge-embedding and recursion point edge-embedding:

Let  $U$  denote the part of  $U_n^{(4)}(\Gamma_1)$  without recursion steps (the main chain of blocks) and  $G_1 = (V, E)$  be the  $\Gamma_1(n)$  graph which is to be edge-embedded in  $U_n^{(4)}(\Gamma_1)$ .  $S$  denotes the set of the 4 subgraphs of  $G_1$  in the supergraph, i.e.  $S = \{G_1^{11}, G_1^{12}, G_1^{21}, G_1^{22}\}$ . A *recursion step graph* of  $U$  is one of the graphs having one of the 4 sets of recursion points as poles (e.g.  $r_1^1, \dots, r_{\lceil \frac{n}{4} \rceil - 1}^1$ ) without the recursion steps.  $R$  denotes the set of all 4 recursion step graphs of  $U$ , and  $B$  denotes the set of all blocks in  $U$ .

We give a brief explanation of Listing 1 that describes the edge-embedding process. For any edge  $e = (v_i, v_j) \in E$  in  $G_1$ ,  $b_i$  and  $b_j$  denote the block numbers in which  $v_i$  and  $v_j$  are. There are 2 cases:



1.  **$v_i$  and  $v_j$  are in the same block:**  $b_i = b_j$ . The edge-embedding can be solved within the block and no recursion points have to be programmed for this path. Therefore, vector *out* of block  $B_{b_i}$  is set accordingly.
2.  **$v_i$  and  $v_j$  are in different blocks:**  $b_i \neq b_j$ . There exists an edge  $e' = (b_i, b_{j-1})$  in one of the four  $\Gamma_1(\lceil \frac{n}{4} - 1 \rceil)$  subgraphs of  $G_1$  that is not yet used for an edge-embedding. This determines that the path in the next recursion step has to be between poles  $p_{b_i}$  and  $p_{b_{j-1}}$ . We denote with  $s \in S$  the subgraph of  $G_1$  which contains  $e'$ , and  $x$  denotes its number in  $S$ , i.e.  $S_x = s$ . This implies in which of the 4 recursion step graphs we need to edge-embed the path from  $p_{b_i}$  to  $p_{b_{j-1}}$ , and so which recursion points we need to program. We first set the programming bit of the  $x$ -th input [output] recursion points to 1 since the path between the poles with labelling  $i$  and  $j$  enters [leaves] the next recursion step over this recursion point. A special case to be considered here is when blocks  $B_{b_i}$  and  $B_{b_j}$  are neighbours (i.e.  $b_j = b_i + 1$ ). Then, the path enters and leaves the next recursion step graph at the same node, whose programming bit thus has to be 0. The output vector of block  $B_{b_i}$  is the  $i'$ th value to the  $x$ th recursion point and the input vector of block  $B_{b_j}$  is the  $x$ th value to the  $j'$ th pole in this block.

We repeat these steps for all edges  $e \in E$ . Since all in- and output vectors of all blocks in  $B$  are set, they can be embedded with the block edge-embedding. For all 4 subgraphs of  $G_1$  in the supergraph and in the EUG, we call the same procedure with  $S_i \in S, R_i \in R, 1 \leq i \leq 4$ .

### 3.2 Edge-Embedding in Lipmaa et al.'s $k$ -Way UC

In this section, we extend the recent work of [LMS16] by providing a detailed and modular embedding mechanism for any  $k$ -way EUG construction described in Sect. 2.5. We provide the main differences to the edge-embedding of the 4-way EUG construction detailed in Sect. 3.1.

**$k$ -Way Block Edge-Embedding.** In this setting, our main block is a programmable block  $B$  of size  $x$  with  $k$  poles  $p_1, \dots, p_k$ , and  $k$  input [output] recursion points  $r_0^1, \dots, r_0^k$  [ $r_1^1, \dots, r_1^k$ ].  $B$  is topologically ordered with mapping  $\eta^U$  as defined in Sect. 2.1. Vectors  $in = (in_1, \dots, in_k) \in \{0, \dots, k\}^k$ , and  $out = (out_1, \dots, out_k) \in \{0, \dots, 2k - 1\}^k$  denote the input and output vectors of  $B$ , respectively. Values  $k, \dots, 2k - 1$  in  $out$  denote the recursion point targets  $r_1^1, \dots, r_1^k$  (cf. Sect. 3.1). We formalize the setting of  $in$  and  $out$  in Eqs. 10–14. We denote with  $\mathcal{P}$  the set of all paths in  $B$ , and the  $\mathcal{P}_B \subseteq \mathcal{P}$  the paths that get edge-embedded in  $B$ .

$$\forall i \in \{1, \dots, k\} : in_i \neq 0 \rightarrow (r_0^i, p_{in_i}) \in \mathcal{P}_B, \quad (10)$$

$$out_i \neq 0 \wedge out_i < k \rightarrow (p_i, p_{1+out_i}) \in \mathcal{P}_B \wedge \eta^U(p_i) < \eta^U(p_{1+out_i}) \quad (11)$$

$$out_i > k - 1 \rightarrow (p_i, r_1^{i-k+1}) \in \mathcal{P}_B. \quad (12)$$

$$\forall in_i, in_j \in in : i \neq j \rightarrow in_i = 0 \vee in_i \neq in_j. \quad (13)$$

$$\forall out_i, out_j \in out : i \neq j \rightarrow out_i = 0 \vee out_i \neq out_j. \quad (14)$$

**$k$ -Way Recursion Point Edge-Embedding.**  $G \in \Gamma_2(n)$  denotes the transformed graph of a Boolean circuit  $C_{u,v}^k$ , where  $n = u + k + v$ .

1. *Splitting*  $G \in \Gamma_2(n)$  in two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ : Similarly as in Sect. 3.1, we first split  $G$  into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$  with 2-coloring.
2. *Merging a  $\Gamma_1(n)$  graph into a  $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$  graph*:  $G_1 = (V, E) \in \Gamma_1(n)$  is merged into a  $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$  graph  $G_m = (V', E')$  (same for  $G_2$ ). Therefore, we redefine mapping  $\theta_V$  (cf. Sect. 3.1) that maps node  $v_i \in V$  to node  $v_j \in V'$ . In this scenario,  $k$  nodes in  $V$  build one node in  $V'$ , so  $\theta_V(v_i) = v_{\lceil \frac{i}{k} \rceil}$ . The mapping of the edges  $\theta_E$  is the same as in the 4-way EUG construction, and  $(v'_i, v'_j) \in E'$  where  $j < i$  edges are removed along with  $v_{\lceil \frac{i}{k} \rceil}$  in the end.  $G_m$  is then a topologically ordered graph in  $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$ .
3. *The supergraph for Lipmaa et al.'s  $k$ -way EUG construction*: The next step is to split  $G_m \in \Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$  into  $k$   $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$  graphs. This is done with  $k$ -coloring: a directed graph  $K = (V, E)$  can be  $k$ -colored, if  $k$  sets  $E_1, \dots, E_k \subseteq E$  cover the following conditions:

$$\forall i, j \in \{1, \dots, k\} : i \neq j \rightarrow E_i \cap E_j = \emptyset. \quad (15)$$

$$\forall e \in E : \exists i \in \{1, \dots, k\} : e \in E_i. \quad (16)$$

$$\begin{aligned} \forall i \in \{1, \dots, k\} : \forall e = (v_1, v_2) \in E_i : \\ \neg \exists e' = (v_3, v_4) \in E_i : v_2 = v_4 \vee v_1 = v_3. \end{aligned} \quad (17)$$

According to König's theorem [Kö31, LP09],  $\Gamma_k(n)$  graphs can always be  $k$ -colored efficiently (cf. full version [GKS17, Appendix A] for details). The rest of the supergraph construction and the way it is used for edge-embedding is the same as for the 4-way EUG construction as described in Sect. 3.1.

**$k$ -Way Edge Embedding Algorithm.** The edge-embedding algorithm is the same as shown in Listing 1, after replacing every 4 with  $k$ .

## 4 New Universal Circuit Constructions

Here, we describe our ideas for novel, potentially more efficient, UC constructions. Firstly, in Sect. 4.1, we describe modular building blocks for a *3-way UC*. We show that Valiant's optimized  $U_3(\Gamma_1)$  cannot directly be applied as a building block in the construction due to the fact that it must have an additional node to be a generic EUG. We prove that the EUG without this node is not a valid EUG by showing a counterexample. Therefore, it actually results in a worse asymptotic size than Valiant's 2-way and 4-way UC constructions. Secondly, in Sect. 4.2, we propose a *hybrid UC construction*, utilizing both Valiant's 2-way and 4-way UC constructions so that the overall size of the resulting hybrid UC is minimized, and is at least as efficient as the better construction for the given size.



### 4.1 3-Way Universal Circuit Construction

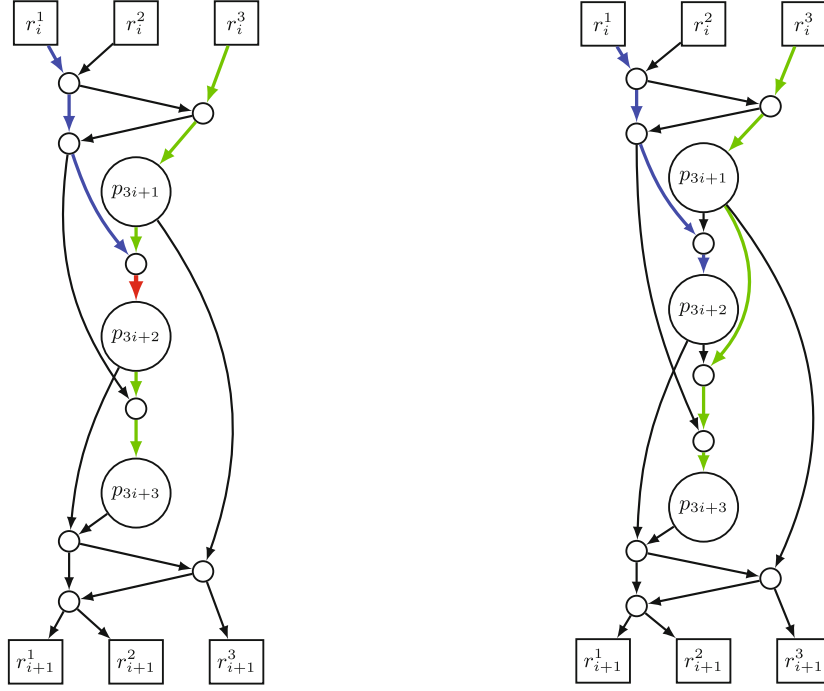
The optimal  $k$  value for minimizing the size of the  $k$ -way UC was calculated to be 3.147 in [LMS16]. We describe our idea of a 3-way UC construction. Intuitively, based on an optimization by Valiant [Val76], this UC should result in the best asymptotic size. The asymptotic size of any  $k$ -way UC depends on the size of its modular body block  $B_k$  (e.g., Fig. 4a for the 4-way UC). Once it is determined, the size of the UC is  $\text{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \text{size}(U_n^{(k)}(\Gamma_1)) \approx 2 \cdot \frac{\text{size}(B_k)}{k} n \log_k n = 2 \cdot \frac{\text{size}(B_k)}{k \log_2(k)} n \log_2 n$ . The modular block consists of two permutation networks  $P(k)$ , an EUG  $U_k(\Gamma_1)$ , and  $(k - 1)$  Y-switching blocks (cf. Sect. 2.5, [LMS16]).

**Size of Body Block  $B_3$  with Valiant's Optimized  $U_3(\Gamma_1)$ .** According to Valiant [Val76], an EUG  $U_3(\Gamma_1)$  with 3 poles contains only 3 connected poles (used as recursion base in Sect. 2.2). An optimal permutation network  $P(3)$  that achieves the lower bound has 3 nodes as well. This implies that  $\text{size}(B_3) = 2 \cdot P(3) + \text{size}(U_3(\Gamma_1)) + (3 - 1) = 11$ . Then, the size of the UC becomes  $\approx 2 \cdot \frac{11}{3 \log_2 3} n \log_2 n \approx 4.627 n \log_2 n$ , which means an asymptotically by around 2.5% smaller size than that of the 4-way UC.

However, there is a flaw in this initial design. Valiant's  $U_3(\Gamma_1)$  only works as an EUG for 3 nodes under special conditions, e.g., when it is a subgraph within a larger EUG construction. There are 3 possible edges in a topologically ordered graph  $G = (V, E)$  in  $\Gamma_1(3)$ :  $(1, 2)$ ,  $(2, 3)$  and  $(1, 3)$ .  $(1, 2)$  and  $(2, 3)$  can be directly embedded in  $U_3(\Gamma_1)$  using  $(p_1, p_2)$  and  $(p_2, p_3)$ , respectively.  $(1, 3)$ , however, has to be embedded as a path *through* node 2, i.e., as a path  $((p_1, p_2), (p_2, p_3))$ . When  $U_3(\Gamma_1)$  is a subgraph of a bigger EUG, this is possible by programming  $p_2$  accordingly. However, when we use this  $U_3(\Gamma_1)$  as a building block in our EUG construction, it cannot directly be applied. A generic  $U_3(\Gamma_1)$  that can embed  $(1, 3)$  without going through  $p_2$  as before has an additional Y-switching block.

We depict in Fig. 5a the 3-way body block that uses Valiant's optimized  $U_3(\Gamma_1)$  in the  $k$ -way block design of [LMS16]. Assume that the output of pole  $p_{3i+1}$  has to be directed to pole  $p_{3i+3}$ . Then, it needs to go through pole  $p_{3i+2}$ , which means that the edge going in to  $p_{3i+2}$  is used by this path. However, there might be an other edge coming from the permutation network as an input to  $p_{3i+2}$ , e.g., from  $p_{3i}$  from the preceding block. This cannot be directed to  $p_{3i+2}$  anymore as shown in Fig. 5a. Therefore, in the 3-way body block construction, it does not suffice to use Valiant's optimized  $U_3(\Gamma_1)$  [Val76].

**Size of Body Block  $B_3$  with Our Generic  $U_3(\Gamma_1)$ .** In Fig. 5b, we show the 3-way body block with the generic  $U_3(\Gamma_1)$  that allows the output from  $p_{3i+1}$  to be directed to  $p_{3i+3}$  without having to go *through*  $p_{3i+2}$ . This results in  $\text{size}(B_3) = 2 \cdot P(3) + \text{size}(U_3(\Gamma_1)) + (3 - 1) = 12$ , which implies that the asymptotic size of the UC is  $\approx 2 \cdot \frac{12}{3 \log_2 3} n \log_2 n \approx 5.047 n \log_2 n$ . Unfortunately, this is worse than the asymptotic size of the 2-way construction, and we therefore conclude that the asymptotically most efficient known UC construction is Valiant's 4-way UC construction.



(a) Body Block with Valiant's  $U_3(\Gamma_1)$     (b) Body Block with our generic  $U_3(\Gamma_1)$

**Fig. 5.** Body block construction for our 3-way EUG  $U_n^{(3)}(\Gamma_1)$ .

## 4.2 Hybrid Universal Circuit Construction

In this section, we detail our hybrid UC that minimizes its size based on Valiant's 2-way and 4-way UCs, which are asymptotically the smallest UCs to date. Given the size of the input circuit  $C_{u,v}^k$ , i.e.,  $n = u + k + v$ , we can calculate at each recursion step if it is better to create 2 subgraphs of size  $\lceil \frac{n}{2} - 1 \rceil$  and utilize the 2-way recursive skeleton, or it is more beneficial to create a 4-way recursive skeleton with 4 subgraphs of size  $\lceil \frac{n}{4} - 1 \rceil$ .

We assume that for every  $n$ , we have an algorithm that computes the size ( $\text{size}(U_n^{\text{hybrid}}(\Gamma_1))$ ) of the hybrid construction for sizes smaller than  $n$ . We give details on how it is computed in Sect. 5. Then, Listing 2 describes the algorithm for constructing a hybrid UC, at each step based on which strategy is more efficient. We note that our hybrid construction is generic, and given multiple  $k$ -way UC constructions as parameter  $K$  ( $K = \{2, 4\}$  in our example), it minimizes the concrete size of the resulting UC.

## 5 Size of UC Constructions

Lipmaa et al.'s  $k$ -way UC construction is depicted in a modular manner in [LMS16, Fig. 12] and discussed briefly in Sect. 2.5 and Fig. 3. They show that a  $k$ -way body block consists of two permutation networks  $P(k)$ , an EUG for  $k$  nodes, i.e.,  $U_k(\Gamma_1)$ , and additionally,  $(k - 1)$  Y-switching blocks. In this section, we recapitulate the sizes (Table 2) of the  $k$ -way EUG and give an estimate for the

**Listing 2.** Hybrid construction algorithm

```

1  procedure hybrid ( $p_1, \dots, p_n, K = \{2, 4\}$ )
2  Let  $\text{size}(U_{n'}^{\text{hybrid}}(\Gamma_1))$  be the function calculating the size of the
    $\hookrightarrow$  smaller hybrid constructions with size  $n' \leq n$ 
3  for all  $k \in K$  do // Number of poles in the last block for all  $k$ 
4    if  $n \mid k$  do
5       $m_k \leftarrow k$ 
6    else
7       $m_k \leftarrow n \bmod k$ 
8    end if
9     $s_k \leftarrow \text{size}(\text{Head}_k(k)) + (\lceil \frac{n}{k} \rceil - 3) \cdot \text{size}(\text{Body}_k(k)) + \text{size}(\text{Body}_k(r_k)) + \text{size}(\text{Tail}_k(m_k)) +$ 
    $\hookrightarrow m_2 \cdot \text{size}\left(\text{size}(U_{\lceil \frac{n}{2} \rceil}^{\text{hybrid}}(\Gamma_1))\right) + ((k - m_k) \cdot \text{size}\left(\text{size}(U_{\lfloor \frac{n}{k} \rfloor}^{\text{hybrid}}(\Gamma_1))\right))$ 
10 end for
11  $s_i \leftarrow \min(s_k : k \in K)$  // Choose the better construction
12 Create skeleton for  $i$ -way construction with  $n$  poles
13 call hybrid( $r_1^1, \dots, r_{\lceil \frac{n}{i} \rceil}^1, K$ ), ..., hybrid( $r_1^{m_i}, \dots, r_{\lceil \frac{n}{i} \rceil}^{m_i}, K$ )
14 if  $(i - m_i) > 0$  do
15   call hybrid( $r_1^{m_i}, \dots, r_{\lfloor \frac{n}{i} \rfloor}^{m_i}, K$ ), ..., hybrid( $r_1^i, \dots, r_{\lfloor \frac{n}{i} \rfloor}^i, K$ )
16 end if
17 end procedure

```

leading constant for Lipmaa et al.'s EUG construction with size  $\mathcal{O}(n \log_2 n)$ , for  $k \in \{2, \dots, 8\}$ . For a detailed discussion on the depth of the UCs, the reader is referred to the full version of this paper [GKS17, Sect. 5]. We conclude that the best asymptotic size is achieved by Valiant's 4-way UC. This result does not exclude the possibility for a more efficient UC in general, but it shows that the most efficient UC using Lipmaa et al.'s  $k$ -way UC from [LMS16] is the 4-way UC. Two  $k$ -way EUGs for  $\Gamma_1(n)$  graphs build up an EUG for  $\Gamma_2(n)$  graphs as described in Sect. 2.1. Therefore, the leading constant for the size of the UC is twice that of the EUG  $U_n^{(k)}(\Gamma_1)$ , which is summarized in the same table.

### 5.1 Asymptotic Size of $k$ -Way UC Constructions

We review the sizes of the building blocks of a  $k$ -way body block, i.e., the size of an EUG  $U_k(\Gamma_1)$  for  $k$ , and the size of a permutation network  $P(k)$  with  $k$  inputs and outputs, as well as the size of the resulting UCs.

**Edge-Universal Graph with  $k$  Poles.** Valiant optimized EUGs up to size 6 by hand in [Val76]: for  $k = 2$ ,  $U_2(\Gamma_1)$  has two poles, for  $k = 3$  we discussed in Sect. 4.1 that an additional node is necessary. For  $k \in \{4, 5, 6\}$  the sizes are  $\{6, 10, 13\}$ , as shown in [KS16, Fig. 1] (note that the nodes noted as empty circles disappear in the UC). For  $k = 7$  and  $k = 8$ , we observe that Valiant's 2-way UC results in a better size than that of the 4-way UC due to the smaller permutation network and less recursion nodes. Therefore, we use these constructions to compute the size of  $U_7(\Gamma_1)$  and  $U_8(\Gamma_1)$ . As mentioned in [LMS16], another possibility is to use the UC of [KS08b] instead of these EUGs since they have better sizes for small circuits. These UCs  $U^{\text{KS08}}(k)$  are built from two smaller  $U^{\text{KS08}}(\frac{k}{2})$ , a  $P(\frac{k}{2})$  and  $\frac{k}{2}$  Y switches. It results in a smaller size of 21 for  $k = 8$ .

**Table 2.** The leading factors of the asymptotic  $\mathcal{O}(n \log_2 n)$  **size** for  $k$ -way edge-universal graphs ( $U_n^{(k)}(\Gamma_1)$ ) and universal circuits (UC) for  $k \in \{2, \dots, 8\}$ .  $n$  denotes the size of the input  $\Gamma_2(n)$  circuit,  $U_k(\Gamma_1)$  the size of Valiant’s edge-universal graph with  $k$  poles,  $U^{\text{KS08}}(k)$  the size of the UC of [KS08b],  $P^l(k)$  the lower bound for the size of a permutation network for  $k$  nodes, and  $P^W(k)$  the size of Waksman’s permutation network [Wak68].  $B_k^W$  is the size of the body block.

$k$	$U_k(\Gamma_1)$	$U^{\text{KS08}}(k)$	$P^l(k)$	$P^W(k)$	$B_k^W$	$U_n^{(k)}(\Gamma_1) (\cdot n \log_2 n)$	UC ( $\cdot n \log_2 n$ )
<b>2</b>	<b>2</b>	2	1	<b>1</b>	5	2.5	5
<b>3</b>	<b>4</b>	6	3	<b>3</b>	12	$\approx 2.524$	$\approx 5.047$
<b>4</b>	<b>6</b>	7	5	<b>5</b>	19	2.375	4.75
<b>5</b>	<b>10</b>	11	7	<b>8</b>	30	$\approx 2.584$	$\approx 5.168$
<b>6</b>	<b>13</b>	14	10	<b>11</b>	40	$\approx 2.579$	$\approx 5.158$
<b>7</b>	<b>19</b>	19	13	<b>14</b>	53	$\approx 2.697$	$\approx 5.394$
<b>8</b>	23	<b>21</b>	16	<b>17</b>	62	$\approx 2.583$	$\approx 5.167$

**Permutation Networks.** Waksman in [Wak68] showed that the lower bound for the size of a permutation network is  $\lceil \log_2(k!) \rceil$  for  $k$  elements. We present this lower bound in Table 2 as  $P^l(k)$ . The permutation network with the smallest size is Waksman’s permutation network  $P^W(k)$  [Wak68, BD02]. For  $k \in \{2, 3, 4\}$  its size reaches the lower bound, but for larger  $k$  values, his permutation network utilizes additional nodes. Since these are the smallest existing permutation networks, we use these when calculating the size of the UC. Even with the lower bound  $P^l(k)$ , for  $k \in \{5, 6, 7, 8\}$  we would have the respective leading terms  $\{4.824, 4.900, 5.190, 5\}$ , which are larger than 4.75 for  $k = 4$ .

**Body Blocks.** A body block  $B_k^W$  is built of  $(k - 1)$  Y-switching blocks, an EUG for  $k$  nodes, and two permutation networks [LMS16] (cf. Fig. 3). The size of  $B_k^W$  is the sum of the sizes of its building blocks, i.e.,  $\text{size}(B_k^W) = \min(\text{size}(U_k(\Gamma_1)), \text{size}(U^{\text{KS08}}(k))) + 2 \cdot \text{size}(P^W(k)) + k - 1$ .

**Edge-Universal Graphs and Universal Circuits with  $n$  Poles.** The asymptotic size of EUG  $U_n^{(k)}(\Gamma_1)$  is determined as  $\text{size}(U_n^{(k)}(\Gamma_1)) = \frac{\text{size}(B_k^W)}{k \log_2 k} n \log_2 n$  and the leading factor for a UC is twice this number.

## 5.2 Concrete Size of UC Constructions

The size of Lipmaa et al.’s  $k$ -way universal circuits depends on the size of their building blocks [LMS16]. More concretely, finding either better edge-universal graphs for small number of nodes or optimal permutation networks could improve the sizes of these UCs. Lipmaa et al. calculated the optimal  $k$  value for minimizing the size of a  $k$ -way UC to be 3.147 [LMS16].

Table 2 shows that the smallest sizes are achieved by the 4-way, followed by the 2-way UCs. The 3-way UC, as mentioned in Sect. 4.1, is less efficient due to

the additional node in  $U_3(\Gamma_1)$ . We observe that the sizes grow with increasing  $k$  values due to the permutation networks and EUGs.

**Concrete Sizes of 4-Way and 2-Way UCs.** Based on the parity (2-way UC) and the remainder modulo 4 (4-way UC), not only the size of the outest skeleton, but also that of the smaller subgraphs can be optimized. It was considered in [KS16] for the 2-way UC, and we now generalize the approach for  $k$ -way UCs. We provide a recursive formula for the concrete size of the optimized  $k$ -way EUG as follows. Let  $m_k$  be defined as

$$m_k := \begin{cases} n \bmod k & \text{if } k \nmid n, \\ k & \text{if } k \mid n. \end{cases} \quad (18)$$

Then, given the designed Head, Body and Tail blocks with sizes shown in Table 3, we can compute the size by calculating the size of all the components of the outest skeleton, and the sizes of the smaller subgraphs with the recursive formula shown in Eq. 19.<sup>4</sup>

$$\begin{aligned} \text{size}(U_n^{(k)}(\Gamma_1)) = & \text{size}(\text{Head}(k)) + \left( \left\lceil \frac{n}{k} \right\rceil - 3 \right) \cdot \text{size}(\text{Body}(k)) \\ & + \text{size}(\text{Body}(m_k)) + \text{size}(\text{Tail}(m_k)) \\ & + m_k \cdot \text{size}\left(U_{\lceil \frac{n}{k} \rceil - 1}^{(k)}(\Gamma_1)\right) + (k - m_k) \cdot \text{size}\left(U_{\lfloor \frac{n}{k} \rfloor - 1}^{(k)}(\Gamma_1)\right). \end{aligned} \quad (19)$$

**Concrete Size of Our Hybrid UC.** We provide a hybrid UC in Sect. 4.2 for minimizing the size of the resulting UC. This construction chooses at each step the skeleton that results in the smallest size and therefore, we provide the recursive algorithm for determining its size in Eq. 20.  $\text{size}(\text{Head}_k(i))$ ,  $\text{size}(\text{Tail}_k(i))$  and

**Table 3.** The sizes of building blocks of the 2-way and 4-way UCs (cf. Figs. 2 and 4).

Block	Head				Body				Tail			
$k \backslash \text{Poles}$	4	3	2	1	4	3	2	1	4	3	2	1
Fig.	-	-	2c	-	-	-	2a	2b	-	-	2d	2e
2-way	-	-	4	-	-	-	5	5	-	-	4	1
Fig.	4e	4g	4h	4i	4a	4b	4c	4d	4f	4g	4h	4i
4-way	14	14	13	12	19	19	18	17	14	9	4	1

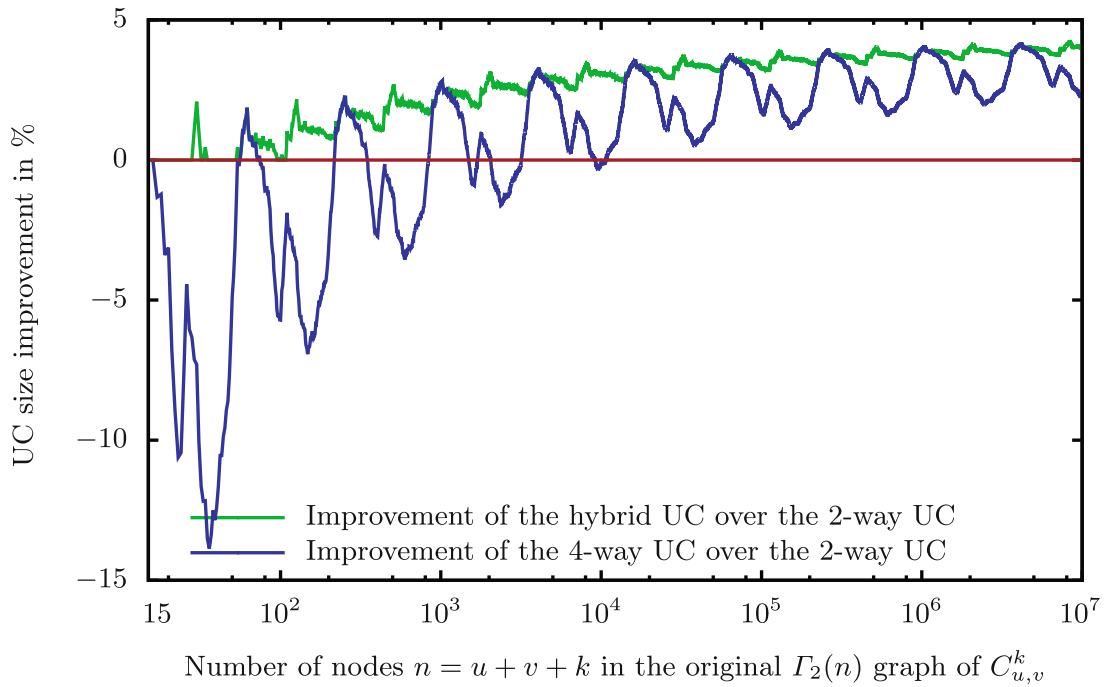
<sup>4</sup> We note that for  $k \geq 3$ , there exist  $\text{Head}(k-1), \dots, \text{Head}(1)$  blocks. These are used for one  $n$ , e.g.,  $\text{Head}(1)$  when  $n = k+1$ , and  $\text{Head}(k-1)$  when  $n = 2k$ . For simplicity, we consider these as special recursion base numbers in our calculations.

$\text{size}(\text{Body}_k(i))$  are the values from Table 3 for  $k = 2$  and  $k = 4$ . The size of the hybrid UC is minimized as

$$\begin{aligned} \text{size}(U_n^{\text{hybrid}}(\Gamma_1)) = & \min \left( \text{size}(\text{Head}_k(k)) + \left( \left\lceil \frac{n}{k} \right\rceil - 3 \right) \cdot \text{size}(\text{Body}_k(k)) \right. \\ & + \text{size}(\text{Body}_k(m_k)) + \text{size}(\text{Tail}_k(m_k)) + m_k \cdot \text{size} \left( U_{\lceil \frac{n}{k} - 1 \rceil}^{\text{hybrid}}(\Gamma_1) \right) \\ & \left. + (k - m_k) \cdot \text{size} \left( U_{\lfloor \frac{n}{k} - 1 \rfloor}^{\text{hybrid}}(\Gamma_1) \right) \right); \quad k \in \{2, 4\}, \end{aligned} \quad (20)$$

which can be computed using a dynamic programming algorithm.

**Improvement of 4-Way Construction.** The bottom (blue) line in Fig. 6 shows the concrete improvement in percentage of the 4-way UC construction over the 2-way UC construction up to ten million nodes in the simulated input circuit. From the asymptotic leading factors in Table 2, we expect an improvement of up to  $1 - \frac{4.75}{5} = 5\%$ . For the smallest  $n$  values ( $n \leq 15$ ), the 2-way UC is up to 33.3% better than the 4-way UC. However, from  $n = 212$  on, the 4-way UC construction is better, except for some short intervals as shown in Fig. 6 (the difference in these intervals, however, is at most 3.45%). From here on, the 4-way UC is on average 3.12% better in our experiments, where the biggest improvement is 4.48%. Moreover, from  $n = 10885$  on, the 4-way UC always outperforms the 2-way UC.



**Fig. 6.** Improvement of our hybrid and Valiant’s 4-way UC over Valiant’s 2-way UC for  $15 \leq n \leq 10^7$  with logarithmic  $x$  axis. (Color figure online)



**Improvement of Hybrid Construction.** The improvement achieved by our hybrid construction (cf. Sect. 4.2) is depicted in the same Fig. 6, as the top (green) line. For some  $n$  values the hybrid UC achieves the same size as the 2- or 4-way UCs, but due to its nature, it is never worse. This means that the improvement of our hybrid UC is always nonnegative, and greater than or equal to the improvement achieved by the 4-way UC. Moreover, in most cases the hybrid UC results in better sizes than any of the other two constructions: this means that some subgraphs are created for an  $n$  for which the 2-way UC is smaller, and therefore the 2-way recursive structure is utilized. The overall improvement for all  $n$  values is on average 3.65% and at most 4.48% over the 2-way UC construction.

## 6 Implementation and Evaluation

The first implementation of Valiant’s 2-way UC, along with a toolchain for PFE (cf. Sect. 1.1) was given in [KS16]. The 4-way UC has smaller asymptotic size  $\sim 4.75n \log_2 n$ , but has not been implemented before due to its more complicated structure and embedding algorithm.

In this work, we improve the implementation of the open-source framework of [KS16] by using the 4-way UC construction that can directly be applied in the PFE framework. Our improved implementation is available at <http://encrypto.de/code/UC>. Firstly, the functionality is translated to a Boolean circuit using the Fairplay compiler [MNPS04, BNP08]. This is then transformed into a circuit in  $\Gamma_2(n)$ , i.e., with at most two incoming and outgoing wires for each gate, input and output. This is done in a preprocessing step of the framework in [KS16]. The input circuit description of our UC implementation is the same as that of the UC compiler of [KS16], and we also adapt our output UC format to that of [KS16] that includes the gate types described in Sect. 2.2. This format is compatible with the ABY framework [DSZ15] for secure function evaluation.

We discuss our implementation of Valiant’s 4-way UC in Sect. 6.1 and give experimental results in Sect. 6.2. For a description on how the hybrid UC can be implemented, the reader is referred to the full version [GKS17, Sect. 6.3].

### 6.1 Our 4-Way Universal Circuit Implementation

The architecture of our UC implementation is the same as that of [KS16], and therefore, we describe our UC design based on the steps described in [KS16, Fig. 6]. Our implementation gets as input a circuit with  $u$  inputs,  $v$  outputs and  $k$  gates, and outputs a 4-way UC with size  $n = u + k + v$ , as well as the programming  $p_f$  corresponding to the input circuit (cf. Sect. 1).

**Transforming Circuit  $C_{u,v}^k$  into  $\Gamma_2(u + k + v)$  Graph  $G$ .** As a first step, we transform the circuit  $C_{u,v}^k$  into a  $\Gamma_2(n)$  graph  $G = (V, E)$  with  $n = u + k + v$  (cf. Sect. 2.1). Then, we define a topological order  $\eta^G$  on the nodes of  $G$  s.t. every input node  $v_i$  has a topological order of  $1 \leq \eta^G(v_i) \leq u$  and every output node  $v_j$  is labelled with  $u + k + 1 \leq \eta^G(v_j) \leq u + k + v$ .

**Table 4.** Comparison of the sizes of the UCs (2-way, 4-way, and hybrid) for sample circuits from [TS15]. Bold numbers denote if the 2-way or the 4-way UC is smaller; the smallest size is always achieved by our hybrid UC. The UC generation time is given for both implemented UCs.

Circuit	$n$	Circuit size (#AND gates)			UC generation (ms)	
		2-way UC [KS16]	Our 4-way UC	Our Hybrid UC	2-way UC [KS16]	Our 4-way UC
AES-non-exp	46 847	$2.96 \cdot 10^6$	<b><math>2.93 \cdot 10^6</math></b>	$2.86 \cdot 10^6$	9 008.9	10 325.8
AES-exp	38 518	$2.38 \cdot 10^6$	<b><math>2.38 \cdot 10^6</math></b>	$2.31 \cdot 10^6$	6 961.7	8 361.3
DES-non-exp	31 946	$1.96 \cdot 10^6$	<b><math>1.92 \cdot 10^6</math></b>	$1.89 \cdot 10^6$	5 563.8	6 599.5
DES-exp	32 207	$19.8 \cdot 10^6$	<b><math>19.4 \cdot 10^6</math></b>	$1.90 \cdot 10^6$	5 654.0	6 765.0
md5	66 497	$4.42 \cdot 10^6$	<b><math>4.26 \cdot 10^6</math></b>	$4.26 \cdot 10^6$	14 805.5	14 897.8
sha-256	201 206	$1.49 \cdot 10^7$	<b><math>1.46 \cdot 10^7</math></b>	$1.44 \cdot 10^7$	81 889.1	57 439.0
add_32	342	$9.58 \cdot 10^3$	<b><math>9.55 \cdot 10^3</math></b>	$9.44 \cdot 10^3$	29.6	35.3
add_64	674	<b><math>2.21 \cdot 10^4</math></b>	$2.27 \cdot 10^4$	$2.17 \cdot 10^4$	53.9	89.6
comp_32	216	<b><math>5.53 \cdot 10^3</math></b>	$5.54 \cdot 10^3$	$5.49 \cdot 10^3$	17.7	21.2
mult_32x32	12 202	$6.54 \cdot 10^5$	<b><math>6.50 \cdot 10^5</math></b>	$6.35 \cdot 10^5$	1 639.2	2 177.1
Branching_18	200	<b><math>4.92 \cdot 10^3</math></b>	$5.07 \cdot 10^3$	$4.88 \cdot 10^3$	21.0	24.2
CreditChecking	82	<b><math>1.50 \cdot 10^3</math></b>	$1.51 \cdot 10^3$	$1.49 \cdot 10^3$	3.1	12.7
MobileCode	160	<b><math>3.65 \cdot 10^3</math></b>	$3.88 \cdot 10^3$	$3.61 \cdot 10^3$	10.6	29.0

**Creating an EUG  $U_n^{(4)}(\Gamma_2)$  for  $\Gamma_2(n)$  Graphs.** An EUG  $U_n^{(4)}(\Gamma_2)$  is constructed by creating two instances of  $U_n^{(4)}(\Gamma_1)$  as shown in Sect. 2.2. The two instances get merged to  $U_n^{(4)}(\Gamma_2)$  so that one builds the left inputs and outputs and the other builds the right inputs and outputs of the gates (based on the two-coloring of  $G$ ). We create the EUGs with Valiant’s 4-way EUG [Val76] with our optimized blocks from Sect. 3.1 (cf. Fig. 4).

**Programming  $U_n^{(4)}(\Gamma_2)$  to Compute  $C_{u,v}^k$ .** We edge-embed graph  $G$  into  $U_n^{(4)}(\Gamma_2)$  as described in Sect. 3.1. [KS16] use their supergraph construction to define the paths between the poles uniquely for Valiant’s 2-way EUG. We modify this supergraph as described in Sect. 3.1 for Valiant’s 4-way EUG and perform the edge-embedding as described in Listing 1. The programming bits of the nodes are set during the edge-embedding process along the paths between the poles. The block edge-embedding is done by analyzing the possible input values and defining the valid paths as described in Sect. 3.1.

**Outputting a Universal Circuit with Its Programming.** As a final step, EUG  $U_n^{(4)}(\Gamma_2)$  is topologically ordered and output in the UC format of [KS16]. The programming bits  $p_f$  defined by the embedding are also output in a separate file based on the topological order.



## 6.2 Our Experimental Results

In order to show the improvement of our method, we ran experiments on a Desktop PC, equipped with an Intel Haswell i7-4770K CPU with 3.5 GHz and 16 GB RAM, and provide our results in Table 4. To compare with the runtime of the UC compiler of [KS16], we ran the same experiments on the same platform using their 2-way UC implementation.

As [KS16], we use a set of real-life circuits from [TS15] for our benchmarks, and compare the sizes of the resulting circuits and the generation and embedding runtimes. We can see that from the 2-way and 4-way UC constructions, the 4-way UC, as expected, is always smaller for large circuits than the 2-way UC. However, it is sometimes better even for small circuits, e.g., for 32-bit addition with  $n = 342$ . The hybrid construction always provides the smallest UC for our example circuits.

In the last two columns, we report the runtime of the UC compiler of [KS16] and our 4-way UC implementation for generating and programming the universal circuit corresponding to the example circuits. Table 4 shows that the differences in runtime are not significant, and due to its more complicated structure, the 4-way UC takes more time to generate and program in general. However, we can see from the largest example with more than 200000 nodes that asymptotically, the 4-way UC results in a runtime improvement as well, as less nodes need to be programmed.

**Acknowledgements.** This work has been co-funded by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP and by the DFG as part of project E3 within CROSSING. We thank the reviewers of ASIACRYPT’17 for their helpful comments.

## References

- [AMPR14] Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-55220-5\\_22](https://doi.org/10.1007/978-3-642-55220-5_22)
- [Att14] Attrapadung, N.: Fully secure and succinct attribute based encryption for circuits from multi-linear maps. Cryptology ePrint Archive, Report 2014/772 (2014). <http://ia.cr/2014/772>
- [BBKL17] Bicer, O., Bingol, M.A., Kiraz, M.S., Levi, A.: Towards practical PFE: an efficient 2-party private function evaluation protocol based on half gates. Cryptology ePrint Archive, Report 2017/415 (2017). <http://ia.cr/2017/415>
- [BD02] Beauquier, B., Darrot, É.: On arbitrary size Waksman networks and their vulnerability. Parallel Proces. Lett. **12**(3–4), 287–296 (2002)

- [BFK+09] Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.-R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 424–439. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-04444-1\\_26](https://doi.org/10.1007/978-3-642-04444-1_26)
- [BNP08] Ben-David, A., Nisan, N., Pinkas, B.: FairplayMP: a system for secure multi-party computation. In: CCS 2008, pp. 257–266. ACM (2008)
- [BOKP15] Banescu, S., Ochoa, M., Kunze, N., Pretschner, A.: Idea: benchmarking indistinguishability obfuscation – a candidate implementation. In: Piessens, F., Caballero, J., Bielova, N. (eds.) ESSoS 2015. LNCS, vol. 8978, pp. 149–156. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-15618-7\\_12](https://doi.org/10.1007/978-3-319-15618-7_12)
- [BPSW07] Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: CCS 2007, pp. 498–507. ACM (2007)
- [DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS 2015. The Internet Society (2015). Code: <http://crypto.de/code/ABY>
- [FAZ05] Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: Electronic Commerce (EC 2005), pp. 147–154. ACM (2005)
- [FGP14] Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: CCS 2015, pp. 844–855. ACM (2014)
- [FVK+15] Fisch, B., Vo, B., Krell, F., Kumarasubramanian, A., Kolesnikov, V., Malkin, T., Bellovin, S.M.: Malicious-client security in blind seer: a scalable private DBMS. In: IEEE S&P 2015, pp. 395–410. IEEE (2015)
- [GGH+13a] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: FOCS 2013, pp. 40–49. IEEE (2013)
- [GGH+13b] Garg, S., Gentry, C., Halevi, S., Sahai, A., Waters, B.: Attribute-based encryption for circuits from multilinear maps. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 479–499. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40084-1\\_27](https://doi.org/10.1007/978-3-642-40084-1_27)
- [GHV10] Gentry, C., Halevi, S., Vaikuntanathan, V.: *i*-hop homomorphic encryption and rerandomizable Yao circuits. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 155–172. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14623-7\\_9](https://doi.org/10.1007/978-3-642-14623-7_9)
- [GKS17] Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. Cryptology ePrint Archive, Report 2017/798 (2017). <http://ia.cr/2017/798>
- [HKK+14] Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-44381-1\\_26](https://doi.org/10.1007/978-3-662-44381-1_26)
- [Kö31] König, D.: Gráfok és mátrixok. Matematikai és Fizikai Lapok **38**, 116–119 (1931)
- [KM11] Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25385-0\\_30](https://doi.org/10.1007/978-3-642-25385-0_30)
- [KR11] Kamara, S., Raykova, M.: Secure outsourced computation in a multi-tenant cloud. In: IBM Workshop on Cryptography and Security in Clouds (2011)

- [KS08a] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70583-3\\_40](https://doi.org/10.1007/978-3-540-70583-3_40)
- [KS08b] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85230-8\\_7](https://doi.org/10.1007/978-3-540-85230-8_7)
- [KS16] Kiss, Á., Schneider, T.: Valiant’s universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49890-3\\_27](https://doi.org/10.1007/978-3-662-49890-3_27)
- [LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant’s universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/017 (2016). <http://ia.cr/2016/017>
- [LP09] Lovász, L., Plummer, M.D.: Matching Theory. AMS Chelsea Publishing Series. American Mathematical Society, Providence (2009)
- [LR15] Lindell, Y., Riva, B.: Blazing fast 2PC in the offline/online setting with security for malicious adversaries. In: CCS 2015, pp. 579–590. ACM (2015)
- [MNPS04] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay - secure two-party computation system. In: USENIX Security 2004, pp. 287–302. USENIX (2004)
- [MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_33](https://doi.org/10.1007/978-3-642-38348-9_33)
- [MSS14] Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-45608-8\\_26](https://doi.org/10.1007/978-3-662-45608-8_26)
- [NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Conference on Electronic Commerce (EC 1999), pp. 129–139. ACM (1999)
- [NSMS14] Niksefat, S., Sadeghiyan, B., Mohassel, P., Sadeghian, S.S.: ZIDS: a privacy-preserving intrusion detection system using secure two-party computation protocols. *Comput. J.* **57**(4), 494–509 (2014)
- [OI05] Ostrovsky, R., Skeith, W.E.: Private searching on streaming data. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 223–240. Springer, Heidelberg (2005). [https://doi.org/10.1007/11535218\\_14](https://doi.org/10.1007/11535218_14)
- [PKV+14] Pappas, V., Krell, F., Vo, B., Kolesnikov, V., Malkin, T., Geol Choi, S., George, W., Keromytis, A.D., Bellovin, S.: Blind seer: a scalable private DBMS. In: IEEE S&P 2014, pp. 359–374. IEEE (2014)
- [Sch08] Schneider, S.: Practical secure function evaluation. Master’s thesis, University Erlangen-Nürnberg, Germany, February 2008
- [Sha49] Shannon, C.: The synthesis of two-terminal switching circuits. *Bell Labs Tech. J.* **28**(1), 59–98 (1949)
- [SS08] Sadeghi, A.-R., Schneider, T.: Generalized universal circuits for secure evaluation of private functions with application to data classification. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 336–353. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00730-9\\_21](https://doi.org/10.1007/978-3-642-00730-9_21)

- [TS15] Tillich, S., Smart, N.: Circuits of basic functions suitable for MPC and FHE (2015). <https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/>
- [Val76] Valiant, L.G.: Universal circuits (preliminary report). In: STOC 1976, pp. 196–203. ACM (1976)
- [Wak68] Waksman, A.: A permutation network. J. ACM **15**(1), 159–163 (1968)
- [Weg87] Wegener, I.: The complexity of Boolean functions. Wiley-Teubner (1987)
- [Yao86] Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS 1986, pp. 162–167. IEEE (1986)
- [Zim15] Zimmerman, J.: How to obfuscate programs directly. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 439–467. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_15](https://doi.org/10.1007/978-3-662-46803-6_15)

## C Efficient and Scalable Universal Circuits (JoC'20)

---

- [AGKS20] M. Y. ALHASSAN, D. GÜNTHER, Á. KISS, T. SCHNEIDER. “**Efficient and Scalable Universal Circuits**”. In: *Journal of Cryptology (JoC)* 33.3 (2020). Springer. Online version: <https://ia.cr/2019/348>. Code: <https://encrypto.de/code/UC>, pp. 1216–1271. CORE Rank A\*. Appendix C.

<https://doi.org/10.1007/s00145-020-09346-z>



## Efficient and Scalable Universal Circuits\*

Masaud Y. Alhassan · Daniel Günther · Ágnes Kiss · Thomas Schneider

Technical University of Darmstadt, Darmstadt, Germany

sophismay@gmail.com

guenther@encrypto.cs.tu-darmstadt.de

kiss@encrypto.cs.tu-darmstadt.de

schneider@encrypto.cs.tu-darmstadt.de

Communicated by Nigel Smart.

Received 28 March 2019 / Revised 18 February 2020

**Abstract.** A universal circuit (UC) can be programmed to simulate any circuit up to a given size  $n$  by specifying its program inputs. It provides elegant solutions in various application scenarios, e.g., for private function evaluation (PFE) and for improving the flexibility of attribute-based encryption schemes. The asymptotic lower bound for the size of a UC is  $\Omega(n \log n)$ , and Valiant (STOC'76) provided two theoretical constructions, the so-called 2-way and 4-way UCs (i.e., recursive constructions with 2 and 4 substructures), with asymptotic sizes  $\sim 5n \log_2 n$  and  $\sim 4.75n \log_2 n$ , respectively. In this article, we present and extend our results published in (Kiss and Schneider EUROCRYPT'16) and (Günther et al. ASIACRYPT'17). We validate the practicality of Valiant's UCs by realizing the 2-way and 4-way UCs in our modular open-source implementation. We also provide an example implementation for PFE using these size-optimized UCs. We propose a 2/4-hybrid approach that combines the 2-way and the 4-way UCs in order to minimize the size of the resulting UC. We realize that the bottleneck in universal circuit generation and programming becomes the memory consumption of the program since the whole structure of size  $\mathcal{O}(n \log n)$  is handled by the algorithms in memory. In this work, we overcome this by designing novel scalable algorithms for the UC generation and programming. Both algorithms use only  $\mathcal{O}(n)$  memory at any point in time. We prove the practicality of our scalable design with a scalable proof-of-concept implementation for generating Valiant's 4-way UC. We note that this can be extended to work with optimized building blocks analogously. Moreover, we substantially improve the size of our UCs by including and implementing the recent optimization of Zhao et al. (ASIACRYPT'19) that reduces the asymptotic size of the 4-way UC to  $\sim 4.5n \log_2 n$ . Furthermore, we include their optimization in the implementation of our 2/4-hybrid UC which yields the smallest UC construction known so far.

**Keywords.** Universal circuit, Private function evaluation, Function hiding, Scalability.

---

\*This article is a combined and substantially extended version of [45] (EUROCRYPT'16) and [31] (ASIACRYPT'17). We summarize the additional contributions in Sect. 1.3.

## 1. Introduction

Any computable Boolean function  $f(x)$  can be represented as a Boolean circuit  $C_{u,v}^g(x)$  with  $u$  input wires  $x = (\text{in}_1, \dots, \text{in}_u)$ ,  $v$  output wires  $\text{out}_1, \dots, \text{out}_v$ , and  $g$  gates for some  $u, v, g$ . The size of such a Boolean circuit is  $n = u + v + g$ . Universal circuits (UCs) are programmable circuits that can simulate any Boolean function  $f(x)$  up to a given size  $n$ . To program a UC to compute  $f$ , programming or control bits are specified as further inputs  $c^f = \{c_1, \dots, c_m\}$ . The UC then receives these control bits as inputs along with the input  $x$  and computes the result as  $UC(x, c^f) = f(x)$ . This means that the same UC can evaluate different Boolean circuits by specifying the respective control bits. In analogy to a universal Turing machine, a universal circuit allows to turn any function into data in the form of a program description.

Several efficient constructions considering both the size and the depth of UCs were proposed. Valiant proposed in [66] an asymptotically size-optimal UC construction with size  $\Theta(n \log n)$  and depth  $\mathcal{O}(n)$  [68]. He presents two constructions, called 2-way and 4-way UCs, based on so-called edge-universal graphs (EUGs) that utilize either 2 or 4 subcircuits, respectively. The asymptotic complexity of the 4-way UC is  $\sim 4.75n \log_2 n$  which is smaller than that of the 2-way UC of  $\sim 5n \log_2 n$  [66]. The 4-way UC has been further improved in [72], where its size is reduced to  $\sim 4.5n \log_2 n$ . An asymptotically depth-optimal construction with depth  $\Theta(d)$  that simulates circuits with depth  $d$  was proposed in [17], but it has a significantly larger size of  $\mathcal{O}(n^3 d / \log n)$ . In our paper, due to the applications in cryptography that we revisit in Sect. 1.1, we concentrate on the existing size-optimized UCs, especially that proposed by Valiant [66] with asymptotic size  $\Theta(n \log n)$  with the optimization presented by Zhao et al. in [72].

### 1.1. Applications of Universal Circuits

Size-optimized universal circuits have many applications, which we review here and refer to the original publications for a more detailed description.

#### *Private Function Evaluation (PFE)*

The most prominent application of universal circuits is the secure evaluation of private functions based on *secure function evaluation* (SFE) or *secure computation*. SFE enables two parties  $P_1$  and  $P_2$  to evaluate a publicly known function  $f(x, y)$  on their respective private inputs  $x$  and  $y$ , ensuring that none of the participants learns anything about the other participant's input apart from the output of the computation. Many secure computation protocols, such as Yao's garbled circuit protocol [47, 69, 70] and the GMW protocol [32], use Boolean circuits for representing the desired functionality. In some applications, the function itself should be kept private. This setting is called *private function evaluation* (PFE), where we assume that only one of the parties  $P_1$  knows the function  $f(x)$ , whereas the other party  $P_2$  provides the input to the private function  $x$ .  $P_2$  should learn no information about  $f$  except for an upper bound on the size of the circuit describing the function, and  $P_1$  should learn nothing about  $x$  beyond what can be inferred from the result  $f(x)$ .



PFE can be reduced to SFE [1, 44, 58, 63] by securely evaluating a UC that is programmed by  $P_1$  to evaluate the function  $f$  on  $P_2$ 's input  $x$ . For this,  $P_1$  provides the control bits  $c^f$  for the UC and  $P_2$  provides his private input  $x$  into an SFE protocol that computes  $UC(x, c^f)$ . Here, the UC is a public function and the control bits  $c^f$ —and therefore the function  $f$ —and input  $x$  are kept private due to the properties of SFE. The first implementation of PFE was provided in [44, 61], which extends the Fairplay secure computation framework [51] with universal circuits. The underlying UC construction achieves a non-optimal asymptotic size of  $\mathcal{O}(n \log^2 n)$  and depth  $\mathcal{O}(n \log n)$ . We have shown in [45] that it results in larger UCs than Valiant's constructions for all reasonable circuit sizes in practice. The complexity of PFE in this case is determined mainly by the size and depth of the UC, while the security follows from that of the SFE protocol that is used to evaluate the UC. If the SFE protocol is secure against semi-honest, covert, or malicious adversaries, then the PFE protocol is secure in the same adversarial setting. UC-based PFE can be easily integrated into any SFE framework and can directly benefit from recent optimizations. For instance, *outsourcing UC-based PFE* to two or multiple servers using XOR secret sharing is directly possible with outsourced SFE [42]. The non-interactive secure computation protocol of [3] can be generalized to obtain a *non-interactive PFE* protocol [46]. Moreover, with UC-based PFE, evaluating public and private parts of a functionality can easily be performed together without modifying the underlying secure computation framework.

In [40], Katz and Malka presented an alternative approach for PFE that does not rely on UCs. They use additively homomorphic public-key encryption as well as a symmetric-key encryption scheme and achieve constant-round PFE with linear  $\mathcal{O}(n)$  communication complexity. However, the number of public-key operations is linear in the circuit size, and due to the gap between the efficiency of public-key and symmetric-key operations, this results in a less efficient protocol. Their protocol is secure against semi-honest adversaries, uses Yao's garbled circuits [70], and has recently been improved in [5], where the authors modify the algorithm to perform one full execution from which information can be reused in subsequent more efficient executions of the protocol. Mohassel and Sadeghian consider PFE with semi-honest adversaries in [53] and propose a generic PFE framework that can be instantiated with different secure computation protocols. Their first protocol uses homomorphic encryption with which they achieve linear complexity  $\mathcal{O}(n)$  in the circuit size  $n$  and their second protocol relies solely on oblivious transfers (OT), which results in a method with  $\mathcal{O}(n \log n)$  symmetric-key operations. The OT-based construction from [53] or PFE using UCs is more desirable than the linear homomorphic encryption-based methods in practice, since using OT extension, the number of expensive public-key operations can significantly be reduced, such that it is independent of the number of OTs [2, 36]. Biçer et al. [6] improve the communication of the OT-based PFE protocol of [53] by around 40%. The asymptotic complexity of the OT-based construction of [53] and Valiant's UCs for PFE is the same, and therefore, we compare these solutions for PFE in more detail in Sect. 8. Mohassel et al. extend the framework from [53] to malicious adversaries in [54] with linear complexity  $\mathcal{O}(n)$ , using additively homomorphic encryption. Active security of UC-based PFE is achieved by using a secure computation protocol with active security. Even though their claimed better efficiency, to the best of our knowledge, these protocols have not yet been imple-



mented and are not as generally applicable as PFE with UCs, e.g., they cannot be easily combined with secure evaluation of public functions.

Semi-private function evaluation (semi-PFE) has been proposed in [60] and allows for PFE where the function  $f$  is in a set of functions  $\mathcal{F}$  known by both parties. This relaxes the necessary topology hiding requirement of generic PFE. Yao's garbled circuit can be used for evaluating circuits of the same topology as shown in [59]. Recently, an automated approach for semi-PFE has been proposed in [39], where the circuits representing  $f \in \mathcal{F}$  have varying topologies, for which a container topology is found that can be programmed to compute any of the available topologies. This has therefore been defined as a set-universal circuit, i.e., a circuit that can be programmed to compute any circuit from a pre-defined set of circuits. This approach has been further improved in [41], where a modified garbled circuit protocol allows for efficient semi-PFE with linear communication in the size of the largest circuit in  $\mathcal{F}$ . However, semi-PFE does not suffice for generic PFE where we have an exponential number of possible circuit topologies.

### *Applications of PFE*

PFE can be applied in scenarios where one of the parties wants to keep the evaluated function private. One of the first applications for PFE was *privacy-preserving checking for credit worthiness* [21], where not only the loanee's data, but also the loaner's function that computes if the loanee is eligible for a credit needs to be kept private. The original scheme, using garbled circuits, can represent simple policies, but by evaluating a UC their scheme can be extended to more complicated credit checking policies. [15] shows an application for secure computation, where evaluating UCs or other PFE protocols would ensure privacy: When *autonomous mobile agents* migrate between several distrusting hosts, the privacy of the inputs of the hosts is achieved using SFE, while privacy of the mobile agent's code can be guaranteed with PFE. [57] shows a method to *filter remote streaming data* obviously, using secret keywords and their combinations. Their scheme can additionally preserve data privacy by using PFE to search the matching data with a private search function. PFE allows for running *proprietary software* on private data, such as privacy-preserving evaluation of *diagnostic programs* that was considered in [13], where the owner of the program does not want to reveal the diagnostic method and the user does not want to reveal his data. Example applications for such programs include medical diagnostics [9] and remote software fault diagnosis, where the function and the user's input are desired to be handled privately. In the protocol presented in [13], the diagnostic programs are represented as binary decision trees or branching programs which can easily be converted into a Boolean circuit representation and evaluated using PFE based on universal circuits. Moreover, PFE can be applied to create *blinded policy evaluation protocols* [20,24]. [20] utilizes UCs for so-called oblivious circuit policies and [18] for hiding the circuit topology in order to create one-time programs. In [25,59], universal circuits are used for hiding *queries in private database management systems* (DBMSs). The Blind Seer DBMS [25] was improved in [59] by making use of a simpler UC for evaluating queries, which does not hide the circuit topology. The authors mention that in case the topology of the SQL formula and the circuit have to be kept private, a generic UC should be utilized. Further applications

of PFE given in [53] are *evaluation of branching programs on encrypted data* [37] and *privacy-preserving intrusion detection* [56].

### *UC Applications Beyond PFE*

Apart from being used for PFE, UCs can be applied in various other scenarios. Efficient *verifiable computation* on encrypted data was studied in [22]. A verifiable computation scheme was proposed for arbitrary computations, and a UC is required to hide the function. [29] make use of UCs for reducing the verifier’s preprocessing step. In [30], a DDH-based *multi-hop homomorphic encryption* scheme is proposed that uses re-randomizable garbled circuits, for which UCs are used to achieve function privacy. When the common reference string is dependent on a function that the verifier is interested in outsourcing, then the function description can be provided as input to a UC of appropriate size. As described in [4], the *Attribute-based encryption* (ABE) schemes [27, 34] for any polynomial-size circuits can be turned into ciphertext-policy ABE by using UCs. The ABE scheme of [28] also uses UCs. Universal circuits can be applied for program obfuscation. Candidates for *indistinguishability obfuscation* are constructed using a UC as a building block in [14, 26]. The algorithm of [26] has been implemented in [12], which can be improved using Valiant’s UC implementation [45]. *Direct program obfuscation* was proposed in [71], where the circuit is a secret key to a UC. [46] mentions that UCs can be applied for secure two-party computation in the batch execution setting, where the cost of evaluating Yao’s garbled circuits is amortized if the same circuit—a UC—is evaluated [35, 49]. This protocol has been made round-optimal in [52].

### *Implied Theoretical Results*

We mention two theoretical results relying on UCs. Both the depth-optimized UC from [17] and Valiant’s size-optimized UCs were adapted in [8] to construct *universal quantum circuits*. The design of *universal parallel computers* was inspired by Valiant’s UCs as well [33, 50].

## 1.2. Our Contributions and Outline

In Sect. 2, we recapitulate the necessary preliminaries for our work. We revisit the asymptotically size-optimal UCs of [66] in Sect. 3. This complex construction makes use of an internal graph representation and programs a so-called edge-universal graph (Sect. 3.1). Thereafter, we describe how an edge-universal graph can be translated into a universal circuit (Sect. 3.2). Finally, we revisit Valiant’s 2-way (Sect. 3.3) and 4-way UCs (Sect. 3.4) and the improved building block proposed by Zhao et al. [72] for the latter.

*Our modular programming algorithm* (Sect. 4). We detail our modular algorithm for programming a universal circuit that provides the description of the input function  $f$  as program bits  $c^f$  to the UC, for both Valiant’s 2-way and 4-way UCs. Our method consists of two steps, the block edge-embedding (Sect. 4.1) and the recursion point edge-embedding (Sect. 4.2).

*New universal circuit constructions and extensions* (Sect. 5). We describe Lipmaa et al.’s generalization [46] of Valiant’s universal circuit to any  $k$ -way UC (Sect. 5.1) and detail how our modular programming algorithm from Sect. 4 can be directly gener-

alized for this extension. We continue with presenting a new 3-way UC (Sect. 5.2) that is predicted to be more efficient than the existing UCs. However, after providing modular building blocks for this UC, we show that it is asymptotically larger than Valiant’s UCs, due to an optimization that cannot be applied for one of its building blocks. Then, we propose a hybrid UC construction (Sect. 5.3) that can efficiently combine  $k$ -way UCs for multiple values of  $k$ . With this, we combine Valiant’s 2-way and 4-way UCs to achieve the smallest universal circuit known so far. Lastly, we provide our scalable algorithms (Sect. 5.4) that allow for generating and programming UCs with only linear  $\mathcal{O}(n)$  memory instead of handling the whole structure of size  $\mathcal{O}(n \log n)$  in memory at once.

*Optimized size and depth of UCs* (Sect. 6). We compare the asymptotic (Sect. 6.1) and concrete (Sect. 6.2) sizes of Valiant’s (2-way and 4-way) UCs and that of different  $k$ -way UCs. We show that of all  $k$ -way UCs of Lipmaa et al. [46], Valiant’s 4-way UC provides the smallest size for large circuits, whereas Valiant’s 2-way UC provides the smallest depth. We include size optimizations, achieving a linear concrete improvement for all UCs. Moreover, we show that our 2/4 hybrid method for generating UCs improves over the 4-way UCs, i.e., both over Valiant’s 4-way UC and over the optimized 4-way UC of [72].

*Implementation of Valiant’s UCs and experiments* (Sect. 7). We detail the steps of our algorithm for a practical realization of Valiant’s UC construction and implement the 2-way and recently optimized 4-way UCs as well as our 2/4 hybrid UC construction. We note that our implementation is the first implementation that includes the optimization of Zhao et al. [72], which achieves the best size  $\sim 4.5n \log_2 n$  to date. We describe the architecture of our UC compiler (Sect. 7.1). We experimentally evaluate the performance of our UC generation and programming algorithms with a set of example circuits (Sect. 7.2). We provide the evaluation of our scalable 4-way UC as well and compare it with our memory-based implementation of Valiant’s 4-way UC.

*Toolchain for private function evaluation using universal circuits* (Sect. 8). We provide the implementation of an example application for universal circuits, namely of private function evaluation (PFE) by extending the ABY secure function evaluation framework [19] to evaluate our universal circuits (Sect. 8.1). We provide the first implementation for PFE with  $\mathcal{O}(n \log n)$  complexity and show experimental results for performing PFE (Sect. 8.2). We theoretically compare PFE with UCs with other state-of-the-art approaches for PFE (Sect. 8.3).

### 1.3. Additions to Conference Versions

This journal article is a significantly extended and improved version of the conference publications [45] and [31]. Our added contributions are as follows.

1. *Optimizations.* We included the optimized building block of [72] in our 4-way and hybrid implementations as well as in the size and depth comparisons. This allows us to compare all state-of-the-art methods for UCs. This is the first implementation of their construction, which has the lowest asymptotic and concrete sizes known so far.

2. *Scalability.* We extend our design and implementation with a scalable 4-way UC construction based on Valiant’s 4-way UC, which reduces the memory complexity from  $\mathcal{O}(n \log n)$  to  $\mathcal{O}(n)$  when generating and programming the universal circuit. This construction involves a novel layer-by-layer approach for generating and topologically ordering the universal circuit and programs the structure according to the recursion steps, i.e., subcircuit by subcircuit.
3. *Universal circuit depths.* We examine the depth of the universal circuits in addition to their sizes, since though being optimized for the latter, some applications also require to minimize the former. For instance, the number of communication rounds in PFE via secure function evaluation with the GMW protocol [32]—which in contrast to Yao’s garbled circuits allows to precompute all symmetric cryptographic operations [64]—depends on the depth of the universal circuit.
4. *Comparison and implementation.* In our previous works, we have compared the 2-way and 4-way UCs with each other and with the only other existing UC of [44]. In this work, we implement the hybrid method that uses both 2-way and 4-way UCs and achieves the best concrete size for all simulated circuit sizes. We also implement our new scalable 4-way UC construction, which utilizes very different algorithms than those applied before for UC generation. We compare these methods with respect to runtime, communication, and memory consumption.

## 2. Preliminaries

As preliminaries for our paper, we introduce the graph and circuit theoretic background in Sect. 2.1 and Sect. 2.2, respectively. We provide a summary of all our notations and abbreviations in “Appendix A.”

### 2.1. Graph Theory

In this section, we describe the graph theoretic preliminaries necessary for our work.

**Definition 1.** The number of incoming [outgoing] edges of a node is called its *indegree* [outdegree]. A graph has *fanin* [fanout]  $\rho$  if the indegree [outdegree] of all its nodes is at most  $\rho$ .

We denote by  $\Gamma_\rho(n)$  the set of all directed acyclic graphs with  $n$  nodes and fanin and fanout  $\rho$ .

**Definition 2.** Let  $G = (V, E)$  be a *directed graph* with set of *nodes*  $V = \{1, \dots, n\}$  and *edges*  $E \subseteq V \times V$ . A mapping  $\eta^G : V \rightarrow \{1, \dots, n\}$  is called *topological order* if  $(i, j) \in E$  implies that  $\eta^G(i) < \eta^G(j)$  and  $\forall i, j \in V : \eta^G(i) = \eta^G(j)$  means that  $i = j$ . In short,  $i > j$  implies that there is no edge or directed path from  $i$  to  $j$ .

A topological order of  $G \in \Gamma_\rho(n)$  can be found with computational complexity  $\mathcal{O}(\rho n)$ . Further on, we require a labeling of the nodes in a topological order.

**Definition 3.** *Edge-embedding* is a mapping from graph  $G = (V, E)$  into  $G' = (V', E')$  that maps  $V$  into  $V'$  one-to-one, with possible additional nodes in  $V'$ , i.e.,  $V \subseteq V'$  and  $E$  into directed paths in  $E'$ , such that all paths are pairwise edge-disjoint, i.e., an edge can be used only in one path.

**Theorem 1.** (Kőnig–Hall theorem) *Given a directed acyclic graph (DAG)  $G \in \Gamma_2(n)$ , the set of edges  $E$  can be separated into two disjoint sets  $E_1$  and  $E_2$ , such that graphs  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$  are instances of  $\Gamma_1(n)$ , having fanin and fanout 1 for each node [38,48,66].*

*Proof of Theorem 1.* Given the set of nodes in topological order  $V = \{1, \dots, n\}$ , we can construct a bipartite graph  $\bar{G} = (\bar{V}, \bar{E})$  with nodes  $\bar{V} = \{m_1, \dots, m_n, m'_1, \dots, m'_n\}$  and edges  $\bar{E}$  such that  $(m_i, m'_j) \in \bar{E}$  if and only if  $(i, j) \in E$ . It is easy to see that the fanin and fanout of the resulting bipartite graph is also 2. The edges of  $\bar{G}$  and thus the corresponding edges of  $G$  can be colored in a way that the result is a valid two-coloring. Having fanin and fanout of at most 2, such coloring can be found directly with the following method:

- 1: **while** there are uncolored edges in  $\bar{G}$  **do**
- 2:   Choose an uncolored edge  $e = (m_i, m'_j)$  randomly and color the path or cycle that contains it in an alternating manner: The neighboring edge(s) of an edge of the first color will be colored with the second color and vice versa.
- 3: **end while**

This edge-coloring can be performed in  $\mathcal{O}(n)$  steps and it defines the edges in  $E_1$  and  $E_2$ , such that  $E_1$  contains the edges colored with color one and  $E_2$  the ones with color two and  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$ .  $\square$

The Kőnig–Hall theorem was used in [45,46] to provide a 2-coloring algorithm for the edges of a graph with fanin and fanout 2. In its originally proposed form, however, Kőnig’s theorem [38,48] applies also for  $k$ -coloring the edges of any graph with at most  $k$  incoming and outgoing edges for each of its nodes. This transformation can be easily generalized to graphs in  $\Gamma_k(n)$ , in which case the resulting bipartite graph will have fanin and fanout  $k$ . We review this theorem and the corresponding algorithm here.

**Theorem 2.** (Kőnig’s theorem) *If  $\bar{G}$  is bipartite and its nodes have at most  $k$  incoming and outgoing edges, then the number of colors sufficient to color all edges of  $\bar{G}$  is  $k$ .*

*Proof of Theorem 2.* ([38,48]) Take colors  $\{1, \dots, k\}$ , and greedily color edges. Let us assume that at some point the coloring stops because we cannot color more edges. In this step,  $(w_i, z_j)$  is an uncolored edge. If we look at the colors of the edges adjacent to  $w_i$  and  $z_j$ , we can define the set of available colors for both nodes. There is at least one color for both  $w_i$  and  $z_j$  due to the fanin and fanout restriction, but there is no color which is available for both nodes, otherwise we could color  $(w_i, z_j)$ .

There is a color that is used in an edge adjacent to  $w_i$ , e.g., color  $a$ , but not on an edge adjacent to  $z_j$ . In the same way, we can find another color  $b$  that is used in an edge



adjacent to  $z_j$ , but not to  $w_i$ . Take the longest unique path  $P$  from  $w_i$  that uses colors  $a$  and  $b$  alternately.

Indirectly, assume that this path also contains  $z_j$ . It then terminates in  $z_j$  due to the fact that  $z_j$  is not adjacent with an edge colored with  $a$ . Then,  $P \cup (w_i, z_j)$  is an odd cycle, which is impossible since  $\overline{G}$  is bipartite. Therefore,  $p$  does not contain  $z_j$ , and we can exchange colors  $a$  and  $b$  on path  $P$  and color  $(w_i, z_j)$  with color  $a$ .

This process is continued until there are no uncolored edges in  $\overline{G}$ .  $\square$

## 2.2. Circuit Theory

**Definition 4.** The *fanin* [*fanout*] of a circuit can be defined analogously to the fanin [*fanout*] of a graph (cf. Definition 1), i.e., the maximum number of incoming [*outgoing*] wires of all its gates, inputs and outputs.

**Theorem 3.** A circuit  $C_{u,v}^{\hat{g}}$  with  $u$  inputs,  $\hat{g}$  gates, and  $v$  outputs and fanin and fanout  $\rho > 2$  can be transformed to a circuit  $C_{u,v}^g$  with fanin and fanout 2.

*Proof of Theorem 3.* Shannon’s expansion theorem [61,62] describes how gates with larger fanin can be reduced to gates with two inputs by adding additional gates, which results in a circuit  $C_{u,v}^{\tilde{g}}$  with  $\tilde{g}$  fanin 2 gates. It was proven in [66] that the general case, where the fanout of the circuit can be any integer  $\rho \geq 2$ , can be transformed to the special case when  $\rho \leq 2$  by introducing copy gates, each of which eliminates one from the extra fanout of the original gate. We place a binary tree in place of each gate with fanout larger than 2, following Valiant’s proposition: „Any gate with fanout  $x + 2$  can be replaced by a binary fanout tree with  $x + 1$  gates” [66, Corollary 3.1]. Thus, the class of Boolean functions with  $u$  inputs and  $v$  outputs that can be realized by acyclic circuits with  $\tilde{g}$  gates and arbitrary fanout can also be realized with an acyclic fanout-2 circuit with  $\tilde{g} \leq g \leq 2\tilde{g} + v$  gates.

**Definition 5.** We can regard  $C_{u,v}^g$  with  $u$  inputs,  $v$  outputs, and  $g$  gates as a  $\Gamma_2(n)$  graph  $G$ —which we commonly refer to as the *graph of circuit*  $C_{u,v}^g$ —with  $n = u + v + g$  by creating a node for each input, gate, and output, and an edge for each wire in  $C_{u,v}^g$ .

## 3. Valiant’s Universal Circuit Constructions

In any circuit  $C_{u,v}^{\hat{g}}$ , the inputs of each of the  $\hat{g}$  gates are either connected to one of the  $u$  inputs, to the output of a previous gate, or are assigned a fixed constant. Due to the nature of Valiant’s edge-universal graph (EUG) construction, the input circuit must have fanin and fanout 2, which can be achieved with the transformations described in Sect. 2.2 and implemented in [44,45]. From here on, and without loss of generality, we assume that our input circuit  $C_{u,v}^g$  has  $u$  inputs,  $g$  gates and  $v$  outputs and fanin and fanout 2.

The size of a function  $f$  represented by a circuit  $C_{u,v}^g$  with fanin and fanout 2 is  $n = u + v + g$ , which can be represented as a graph  $G \in \Gamma_2(n)$ . In this section, we describe Valiant’s UC constructions [66,68] that can be programmed to evaluate

any function of size  $n$ . We explain the general idea behind Valiant's UC construction [66] in Sects. 3.1 and 3.2, and the 2-way and 4-way UCs along with improvements of [31,45,46,72] in Sects. 3.3 and 3.4, respectively.

### 3.1. Valiant's Edge-Universal Graph Construction

Valiant's UC construction relies on the notion of so-called edge-universal graphs that are then translated to universal circuits [66].

**Definition 6.** A graph  $U_n(\Gamma_\rho) = (V_U, E_U)$  is an *edge-universal graph* (EUG) for  $\Gamma_\rho(n)$  if every graph  $G = (V, E)$  in  $\Gamma_\rho(n)$  can be edge-embedded (cf. Definition 3) into  $U_n(\Gamma_\rho)$ .

An EUG  $U_n(\Gamma_\rho)$  has distinguished nodes called *poles*  $P = \{p_1, \dots, p_n\} \subseteq V_U$  where each node  $a \in V = \{1, \dots, n\}$  is mapped to exactly one pole with an injective mapping  $\varphi^V : V \rightarrow V_U$ . This mapping is defined by a concrete topological order  $\eta^G$  of the original graph  $G$  with  $\varphi^V(a) = p_{\eta^G(a)}$ , i.e., every node in  $G$  has a corresponding pole in  $U_n(\Gamma_\rho)$ . Apart from the poles,  $U_n(\Gamma_\rho)$  might have additional nodes that enable the edge-embedding (cf. Sect. 2.1). For each edge  $(a_i, a_j) \in E$ , we then define a path of variable length  $z$  between the corresponding poles  $\varphi^V(a_i) = p_{\eta^G(a_i)} = b_1$  and  $\varphi^V(a_j) = p_{\eta^G(a_j)} = b_z$  as  $(b_1, \dots, b_z)$ , where  $b_1, \dots, b_z \in V_U$ . All these paths are edge-disjoint, i.e., they do not use any edge in  $U_n(\Gamma_\rho)$  in more than one path (cf. Sect. 2.1).

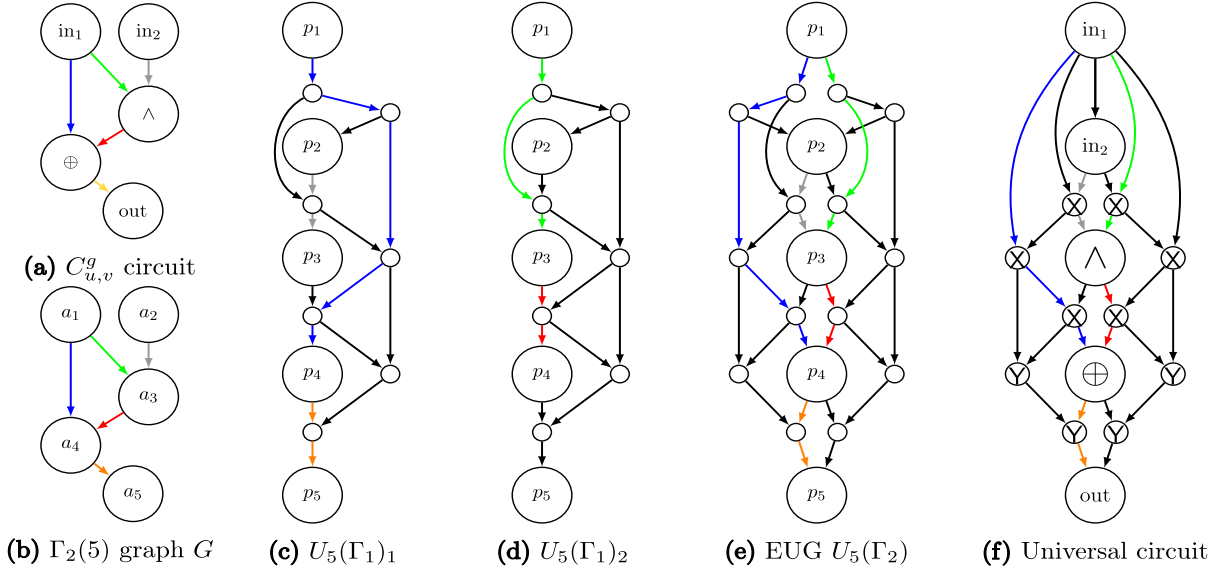
Let  $U_n(\Gamma_1)$  be an EUG for graphs in  $\Gamma_1(n)$  with  $n$  poles  $P = \{p_1, \dots, p_n\}$  (we will show concrete constructions for such EUGs in Sect. 3.3 and in Sect. 3.4). The nodes of any topologically ordered  $\Gamma_1(n)$  graph can be mapped to these poles. The poles have fanin and fanout 1, while all other nodes have fanin and fanout 2.

An EUG  $U_n(\Gamma_\rho)$  for  $\rho \geq 2$  is created by taking  $\rho$  instances of  $U_n(\Gamma_1)$  EUGs with poles  $P_1 = \{p_{1,1}, \dots, p_{1,n}\}, \dots, P_\rho = \{p_{\rho,1}, \dots, p_{\rho,n}\}$ , and merging each pole with its multiple instances, i.e., the set of merged poles  $P = \{p_1, \dots, p_n\}$  is formed by merging  $p_{1,1}, \dots, p_{\rho,i}$  to obtain  $p_i$  for  $i = 1, \dots, n$ . All edges are preserved, and thus, the poles have fanin and fanout  $\rho$ , i.e.,  $U_n(\Gamma_\rho) = (V'_U, E'_U)$  is an EUG with fanin and fanout  $\rho$ , constructed with  $U_n(\Gamma_1)_1 = (V_1, E_1), \dots, U_n(\Gamma_1)_\rho = (V_\rho, E_\rho)$ .  $P$  contains the merged poles and  $V'_U = P \cup_{i=1}^\rho V_i \setminus P_i$  and  $E'_U = \cup_{i=1}^\rho E_i$ . Thus, the poles in  $U_n(\Gamma_\rho)$  have at most  $\rho$  inputs and outputs, and all other nodes have at most two inputs and outputs.

*Example.* Let  $C$  be the circuit shown in Fig. 1a, and  $G = (V, E)$  be the graph of circuit  $C$  with 5 nodes shown in Fig. 1b. Our aim is to edge-embed  $G$  into EUG  $U_5(\Gamma_2)$ . Therefore, we use two instances of  $U_5(\Gamma_1)$ :  $U_5(\Gamma_1)_1$  in Fig. 1c and  $U_5(\Gamma_1)_2$  in Fig. 1d. The edges  $(a_1, a_4)$ ,  $(a_2, a_3)$  and  $(a_4, a_5)$  are embedded in  $U_5(\Gamma_1)_1$ , and the edges  $(a_1, a_3)$  and  $(a_3, a_4)$  in  $U_5(\Gamma_1)_2$ . Merging the poles of  $U_5(\Gamma_1)_1$  and  $U_5(\Gamma_1)_2$  produces  $U_5(\Gamma_2)$  shown in Fig. 1e. In Sect. 3.2, we describe how to retrieve the resulting universal circuit depicted in Fig. 1f.

*Recursion Base.* Valiant's construction is recursive, and the recursion base graphs for up to 6 nodes are shown in [66, Fig. 3] and [45, Fig. 1].  $U_1(\Gamma_1)$  is a single pole and  $U_2(\Gamma_1)$  and  $U_3(\Gamma_1)$  are two- and three-connected poles, respectively. Valiant provides hand-optimized EUGs for  $U_4(\Gamma_1)$ ,  $U_5(\Gamma_1)$  and  $U_6(\Gamma_1)$ , with 3, 7, and 9 additional nodes, respectively (cf. [66, Fig. 3]).





**Fig. 1.** **a** An example circuit and **b** the corresponding  $\Gamma_2(5)$  graph  $G$ . **c**, **d** The edge-embedding of  $G$  into two  $U_5(\Gamma_1)$  instances with poles  $(p_1, \dots, p_5)$ . **e** The edge-embedding of  $G$  into the  $U_5(\Gamma_2)$  graph of the universal circuit shown in **(f)**.

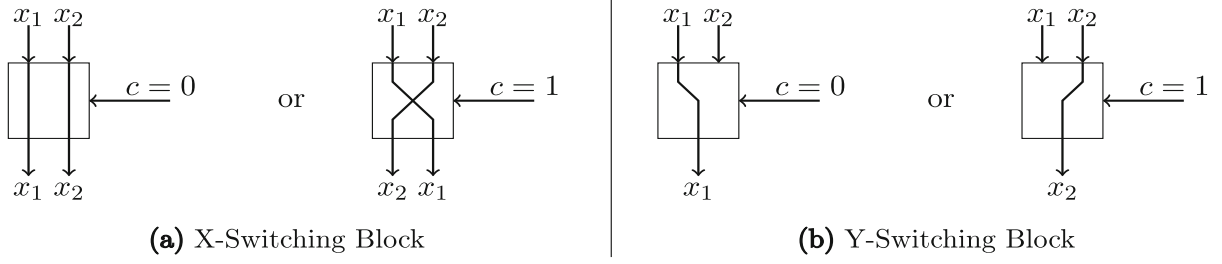
### 3.2. Translating Edge-Universal Graphs into Universal Circuits

In this section, we define universal circuits (UCs) and describe how an edge-universal graph is translated into a universal circuit.

**Definition 7.** A universal circuit  $UC$  is a Boolean circuit that can be programmed to compute any circuit  $C_{u,v}^g$  up to a given size  $n$  by defining a set of programming bits  $c^f$  such that  $UC(x, c^f) = C_{u,v}^g(x)$ .

In Valiant's UC constructions, every node  $w \in V_U$  fulfills a task when  $U_n(\Gamma_2)$  is translated to a UC. Programming the UC means specifying its control bits along the paths defined by the edge-embedding and by the gates of circuit  $C_{u,v}^g$ . Depending on the number of incoming and outgoing edges and its type, a node  $w$  is translated as described below and shown in the example in Fig. 1f.

- G1** If  $w$  is a pole and corresponds to an input (one of the first  $u$  poles) or an output (one of the last  $v$  poles) in  $G$ , then  $w$  is an *input or output* in  $C_{u,v}^g$  as well.
- G2** If  $w$  is not a pole and has indegree 1 and outdegree 2, this node has been placed to copy its input to its two outputs. Therefore, when translated to a UC,  $w$  is replaced by multiple outgoing wires in the parent node (as described in [45]), since the UC does not need to fulfill the fanout 2 restriction. In  $U_n(\Gamma_2)$ ,  $w$  is added due to the fanout 2 restriction in the EUG necessary for the edge-embedding.
- G3** If  $w$  is not a pole and has indegree and outdegree 1,  $w$  is removed and replaced by a wire between its parent and child nodes.
- G4** If  $w$  is a pole and corresponds to a gate (poles  $\{u + 1, \dots, u + g\}$ ) in  $G$ ,  $w$  is programmed as a *universal gate* (UG). A 2-input UG supports any of the 16 possible gate types represented by 4 control bits of the gate table  $(c_1, c_2, c_3, c_4)$ .



**Fig. 2.** Programmable switching blocks [43].

It implements function  $U: \{0, 1\}^2 \times \{0, 1\}^4 \rightarrow \{0, 1\}$  that computes

$$U(x_1, x_2, c_1, c_2, c_3, c_4) = \overline{x_1} \overline{x_2} c_1 + \overline{x_1} x_2 c_2 + x_1 \overline{x_2} c_3 + x_1 x_2 c_4. \quad (1)$$

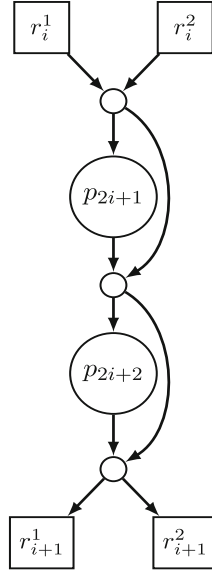
- G5** If  $w$  is not a pole and has indegree and outdegree 2,  $w$  is programmed as an *X-switching block*, which computes  $X: \{0, 1\}^2 \times \{0, 1\} \rightarrow \{0, 1\}^2$  with  $X((x_1, x_2), c) = (x_{1+c}, x_{2-c})$  as shown in Fig. 2a. The inputs of an X-switching block are forwarded to its outputs, switched or not switched, depending on control bit  $c$ .
- G6** If  $w$  is not a pole and has indegree 2 and outdegree 1,  $w$  is programmed as a *Y-switching block* that computes  $Y: \{0, 1\}^2 \times \{0, 1\} \rightarrow \{0, 1\}$  with  $Y((x_1, x_2), c) = x_{1+c}$  as visualized in Fig. 2b. The inputs of a Y-switching block are forwarded to its output depending on the control bit  $c$ , i.e., it provides the functionality of a 2-input multiplexer.

We note that the  $u$  inputs and the  $v$  outputs can be ordered arbitrarily within themselves as long as the inputs are kept before the  $g$  topologically ordered gates and the outputs after them. Even though the output nodes cause an overhead in Valiant's UC, they are required to fully hide the topology of the circuit in the corresponding universal circuit. Note that optionally it is possible to modify the input circuit such that the outputs of the last  $v$  gates in order are the outputs of the circuit by inserting at most  $v$  copy gates [40].

The nodes programmed as UG (G4), X-switching block (G5), or Y-switching block (G6) are so-called *programmable blocks*. This means that a control bit  $c$  or vector  $\bar{c} = (c_1, c_2, c_3, c_4)$  is necessary aside from the two inputs to define their behavior. The universal gates are programmed according to the simulated gates in  $C_{u,v}^g$  and the universal switches according to the paths defined by the edge-embedding of the graph of the circuit  $G$  into the edge-universal graph  $U_n(\Gamma_2)$ . Depending on whether the path takes the same direction during the embedding (e.g., arrives from the left and continues on the left) or changes its direction at a given node (e.g., arrives from the left and continues on the right), the control bit of the universal switch is programmed accordingly. In Sect. 7.1, we describe efficient implementations of programmable blocks. All control bits and vectors together are the programming  $c^f$  of the UC.

### 3.3. Valiant's 2-way UC Construction

We described in Sect. 3.1 that a  $U_n(\Gamma_\rho)$  EUG can be constructed of  $\rho$  instances of  $U_n(\Gamma_1)$  EUGs. Valiant [66] provides an EUG for  $\Gamma_1(n)$  graphs, two of which can

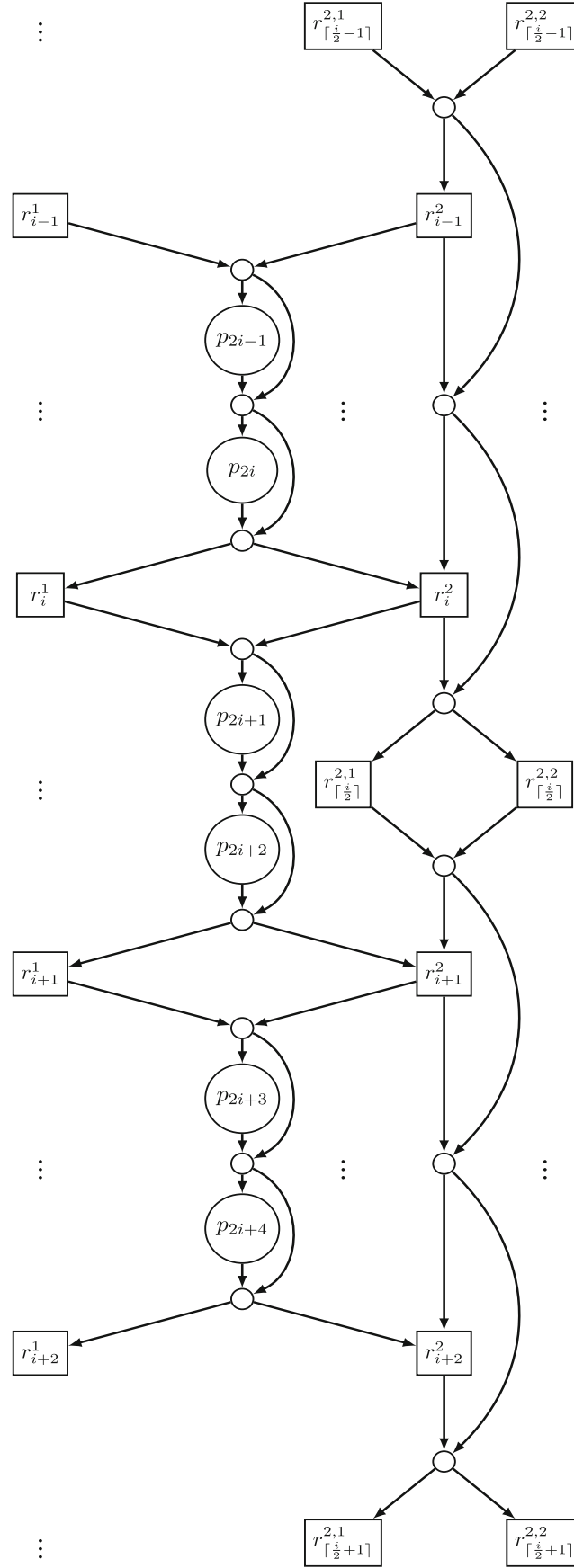


**Fig. 3.** Body block  $B^{(2)}$  of Valiant's 2-way EUG  $U_n^{(2)}(\Gamma_1)$  [66].

build an EUG for  $\Gamma_2(n)$  graphs, which suffices for circuits with 2-input gates that have at most two outgoing wires. Let  $P = \{p_1, \dots, p_n\}$  be the set of poles in  $U_n(\Gamma_1)$  that have indegree and outdegree 1, corresponding to the inputs, gates and outputs of the input circuit  $C_{u,v}^g$ , i.e., poles  $P_{\text{in}} = \{p_1, \dots, p_u\}$  correspond to the inputs,  $P_{\text{gate}} = \{p_{(u+1)}, \dots, p_{(u+g)}\}$  to the gates,  $P_{\text{out}} = \{p_{(u+g+1)}, \dots, p_n\}$  to the outputs. The main, so-called *body block*  $B^{(2)}$  used for constructing Valiant's EUG for  $\Gamma_1(n)$  graphs  $U_n^{(2)}(\Gamma_1)$  of size  $\sim 2.5n \log_2 n$  is shown in Fig. 3 and consists of 2 poles (large circles), 4 so-called recursion points (rectangles), and 3 additional nodes (small circles). The corresponding UC has twice the size  $\sim 5n \log_2 n$ , since it corresponds to an EUG for  $\Gamma_2(n)$  graphs. This construction is called the *2-way EUG or UC construction* since there are two sets of recursion nodes at each recursion step as we describe below.

The recursive construction works as follows: The rectangles are special nodes that build up the set of poles in the next recursion step, i.e.,  $R_{\lceil \frac{n}{2} - 1 \rceil}^1 = \{r_1^1, \dots, r_{\lceil \frac{n}{2} - 1 \rceil}^1\}$  and  $R_{\lceil \frac{n}{2} - 1 \rceil}^2 = \{r_1^2, \dots, r_{\lceil \frac{n}{2} - 1 \rceil}^2\}$  are the poles of two smaller edge-universal graphs called subgraphs. EUGs are built with these poles which produce new subgraphs with size  $\lceil \frac{\lceil \frac{n}{2} - 1 \rceil}{2} - 1 \rceil$ , such that we have four subgraphs at the next level, etc. The blocks are chained together at the recursion points to form a skeleton, i.e., each recursion point belongs to two in the corresponding subgraph. Thus, the main skeleton of the UC consists of  $\lceil \frac{n}{2} \rceil$  such blocks with poles  $\{p_1, p_2, \dots, p_n\}$ , and the next two skeletons consist of  $\lceil \frac{\lceil \frac{n}{2} - 1 \rceil}{2} \rceil$  blocks with sets of poles  $\{r_1^1, \dots, r_{\lceil \frac{n}{2} - 1 \rceil}^1\}$  and  $\{r_1^2, \dots, r_{\lceil \frac{n}{2} - 1 \rceil}^2\}$ . We visualize the process of chaining the blocks together to form this skeleton in Fig. 4.

We note that the top (resp. bottom) block of a skeleton does not need the upper (resp. lower) recursion points since its poles are the inputs (resp. outputs) in the block. Therefore, we presented optimized so-called head  $H^{(2)}$  and tail  $T^{(2)}$  blocks that occur in the top and bottom of a skeleton, respectively, in [31, Fig. 2b–e].



**Fig. 4.** Skeleton built of a chain of body blocks  $B^{(2)}$  of Valiant's 2-way EUG  $U_n^{(2)}(\Gamma_1)$ .

**Theorem 4.** ([66]) *The resulting 2-way EUG is edge-universal, and therefore, the resulting circuit is universal.*

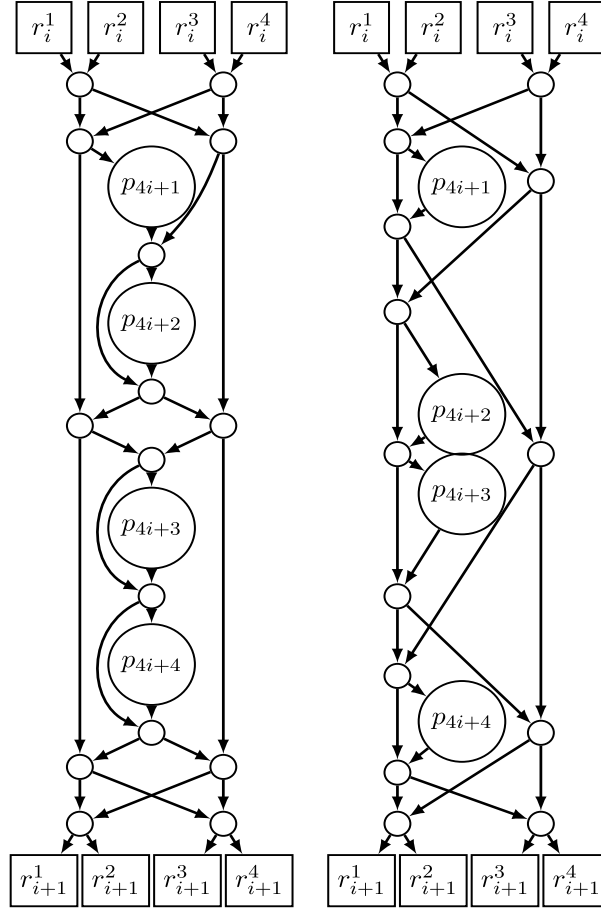
*Proof of Theorem 4 [Val76].* We recapitulate the proof from [66] that  $U_n^{(2)}(\Gamma_1)$  is edge-universal for  $\Gamma_1(n)$ , such that any graph with  $n$  nodes and fanin and fanout 1 can be edge-embedded into  $U_n^{(2)}(\Gamma_1)$ . According to the definition of edge-embedding, it has to be shown that given any  $\Gamma_1(n)$  graph  $G = (V, E)$ , for any  $(i, j) \in E$  and  $(k, l) \in E$  we can find pairwise edge-disjoint paths from  $p_i$  to  $p_j$  and from  $p_k$  to  $p_l$  in  $U_n^{(2)}(\Gamma_1)$ . As before, the labeling of nodes  $V = \{1, \dots, n\}$  in  $G$  is according to a topological order of the nodes.

Firstly, each two neighboring poles of the EUG,  $p_{2s}$  and  $p_{2s+1}$  for  $s \in \{1, \dots, \lceil \frac{n}{2} \rceil\}$ , are thought of as merged poles, so-called superpoles, with their fanin and fanout becoming 2. In a similar manner, any  $G \in \Gamma_1(n)$  graph can be regarded as a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph with supernodes, i.e., each pair  $(2s, 2s+1)$  will be merged into one node in a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph  $G' = (V', E')$ . If there are edges between the nodes in  $G$ , they are simulated with loops. The set of edges of this graph  $G$  is partitioned to disjoint sets  $E_1$  and  $E_2$ , such that  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$  are instances of  $\Gamma_1(\lceil \frac{n}{2} \rceil)$  and  $\Gamma_1(\lfloor \frac{n}{2} \rfloor)$ , respectively. This can be done efficiently, as shown in Theorem 1. The edges in  $E_1$  are embedded as directed paths in  $R_{\lceil \frac{n}{2} - 1 \rceil}^1$ , and the edges in  $E_2$  as directed paths in  $R_{\lceil \frac{n}{2} - 1 \rceil}^2$ . Both  $E_1$  and  $E_2$  have at most one edge directed into and at most one directed out of any supernode, and therefore, there is only one edge from  $E_1$  and one from  $E_2$  to be simulated going through any superpole in  $U_n^{(2)}(\Gamma_1)$  as well. Thus, the edge coming into a superpole  $(p_{2s}, p_{2s+1})$  in  $E_1$  is embedded as a path through  $r_{s-1}^1$ , while the edge going out of the pole in  $E_1$  is embedded as a path through  $r_s^1$  in the appropriate subgraph. Similarly, the edges in  $E_2$  are simulated as edges through  $r_{s-1}^2$  and  $r_s^2$ . These paths can be chosen disjoint according to the induction hypothesis. Finally, the paths from  $r_{s-1}^1$  and  $r_{s-1}^2$  to superpole  $(p_{2s-1}, p_{2s})$  as well as the paths from  $(p_{2s-1}, p_{2s})$  to  $r_s^1$  and  $r_s^2$  can be chosen edge-disjoint due to the skeleton built up of the body blocks shown in Fig. 3. With this, Valiant's graph construction results in a valid EUG with asymptotically optimal size  $\mathcal{O}(n \log n)$  and depth  $\mathcal{O}(n)$  [66]. With the building blocks described in Sect. 3.2, it is easy to see that the resulting Boolean circuit is universal.  $\square$

*Implementation.* We provided an open-source implementation of this 2-way UC optimized for PFE in [45]. In concurrent and independent related work, Lipmaa et al. [46] also showed the practicality of Valiant's 2-way UC. They decrease its total number of gates compared to that of Valiant's block (Fig. 3) by one XOR gate. However, the number of AND gates is exactly the same, and therefore, their improvement does not affect PFE using UCs, where XOR gates are evaluated for free [44].

### 3.4. Valiant's 4-way UC Construction

Similarly to the 2-way EUG construction (cf. Sect. 3.3), Valiant provides a more efficient 4-way EUG or UC construction [66] for  $\Gamma_1(n)$  graphs which can be extended to an EUG for  $\Gamma_2(n)$  graphs by utilizing two instances  $U_n^{(4)}(\Gamma_1)_1$  and  $U_n^{(4)}(\Gamma_1)_2$  as described in Sect. 3.1.  $U_n^{(4)}(\Gamma_1)$  has a 4-way recursive structure, i.e., at each recur-



(a) Body block of Valiant [Val76]. (b) Body block of Zhao et al. [ZYZL19].

**Fig. 5.** Body block  $B^{(4)}$  alternatives for 4-way EUG  $U_n^{(4)}(\Gamma_1)$ .

sion step, nodes in special sets  $R_{\lceil \frac{n}{4} - 1 \rceil}^1 = \{r_1^1, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^1\}$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^2 = \{r_1^2, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^2\}$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^3 = \{r_1^3, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^3\}$  and  $R_{\lceil \frac{n}{4} - 1 \rceil}^4 = \{r_1^4, \dots, r_{\lceil \frac{n}{4} - 1 \rceil}^4\}$ <sup>1</sup> are the poles in the next recursion step (the main body block is shown in Fig. 5a). The recursion base is the same as for the 2-way UC construction described in Sect. 3.1. This construction results in UCs of smaller size  $\sim 4.75n \log_2 n$  but has a more complicated structure and programming algorithm. We have studied and implemented this universal circuit in [31] and recapitulate our results here and in Sect. 7. Valiant offers the main, so-called *body block*  $B^{(4)}$  consisting of 4 poles (large circles), 15 nodes (small circles) as well as 8 recursion points (rectangles) shown in Fig. 5a. As before, we provide so-called *head*  $H^{(4)}$  and *tail*  $T^{(4)}$  blocks that occur at the top and bottom of a skeleton in [31, Figs. 4b-4i], respectively. The blocks are connected such that the 4 top (resp. bottom) recursion points of one block are the 4 bottom (resp. top) recursion points of the next block. Similarly to the 2-way EUG, 4 sets are created for  $n$  nodes, i.e.,  $R_{\lceil \frac{n}{4} - 1 \rceil}^1$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^2$ ,  $R_{\lceil \frac{n}{4} - 1 \rceil}^3$ , and  $R_{\lceil \frac{n}{4} - 1 \rceil}^4$  which are the poles of 4  $U_{\lceil \frac{n}{2} \rceil - 1}(\Gamma_1)$  EUGs in the next recursion step. Then, these also create 4 subgraphs until the recursion base is reached (cf. Sect. 3.1).

<sup>1</sup>  $n \pmod{4}$  of these have size  $\lfloor \frac{n}{4} - 1 \rfloor$ , but for the sake of simplicity, we do not distinguish these here.

Recently, Zhao et al. in [72] optimized the body block of Valiant’s UC by finding a more efficient block using exhaustive search over all possible blocks. As opposed to Valiant’s UC that uses 15 additional nodes in the body block, their block uses only 14 additional nodes, and therefore, their UC achieves an asymptotically better size of  $\sim 4.5n \log_2 n$ . We depict the further optimized body block  $B^{(4)}$  of Zhao et al. in Fig. 5b. Zhao et al. provide a computer generated proof of that this block can indeed be used to construct universal circuits. Moreover, they show that there exists no block with only 13 additional nodes that can be used to construct UCs in the same manner. This proves that the minimal size of a 4-way UC is the achieved  $\sim 4.5n \log_2 n$ .

**Theorem 5.** ([66]) *The resulting 4-way EUG is edge-universal, and therefore, the resulting circuit is universal.*

The proof of this theorem is analogous to that of Theorem 4.

#### 4. Programming Valiant’s Universal Circuits

We designed the detailed embedding algorithm and the open-source UC implementation of [45] specifically for the 2-way UC, dealing with the whole UC skeleton as one block. In contrast, based on the modular design of [46], we modularized the edge-embedding task into multiple subtasks and described how they can be performed separately in [31]. In this section, we detail this modular approach for edge-embedding a graph into Valiant’s  $\ell$ -way EUG, where  $\ell = 2$  or  $\ell = 4$ : The edge-embedding can be split into two parts, which are then combined.

In the following, we describe the two main steps of our modular approach presented in [31] that are based on the edge-embedding algorithm of [45]. 1) Block edge-embedding (Sect. 4.1) allows for the programming of the blocks visualized in Fig. 3 on p. 12 and in Figs. 5a or b on p. 14. 2) Recursion point edge-embedding (Sect. 4.2) takes care of the programming of the whole UC. Here, the paths are defined and the necessary information is provided to the blocks (cf. Sect. 4.2). The process can be generalized to any  $2^i$ -way EUG. Moreover, the same modular edge-embedding algorithm can be applied with a few modifications for Lipmaa et al.’s generalization to any  $k$ -way UC [46], which we describe later in Sect. 5.1.

##### 4.1. Block Edge-Embedding

We consider the  $\ell$  top (resp. bottom) recursion points of a block (Figs. 3 and 5a or b) as intermediate nodes where the inputs (resp. outputs) of the block enter (resp. exit). The blocks are built so that any of these inputs can be forwarded to exactly one of the  $\ell$  poles of the block and the output of any pole can be forwarded to an output or another pole with a higher topological order.

We formalize this behavior as follows: In  $U_n^{(\ell)}(\Gamma_1) = (V_U, E_U)$ , let  $B^{(\ell)}$  be the  $(i - 1)$ th block in the skeleton made up of blocks visualized in Fig. 3 for  $\ell = 2$  and Fig. 5a or b for  $\ell = 4$  with poles  $p_{\ell i+1}, \dots, p_{\ell i+\ell}$ . Let the mapping  $\eta^U : V_U \rightarrow \mathbb{N}^+$  denote a topological order of all nodes and poles in  $V_U$ . Then, the nodes  $r_i^1, \dots, r_i^\ell$  and



$r_{i+1}^1, \dots, r_{i+1}^\ell$  denote the input and output recursion points of block  $B^{(\ell)}$ . Additionally, let  $in = (in_1, \dots, in_\ell) \in \{0, \dots, \ell\}^\ell$  and  $out = (out_1, \dots, out_\ell) \in \{0, \dots, 2\ell - 1\}^\ell$  denote the input and output vectors of  $B^{(\ell)}$ . The value 0 of the input and output vectors is a *dummy value* which is used if there is no specific path between an input and a pole, or between a pole and an output of  $B^{(\ell)}$ . The output vector has a larger value range, since a pole can be forwarded to another pole or an output recursion point. Therefore, we use values  $1, \dots, \ell - 1$  for poles  $p_{\ell i+2}, \dots, p_{\ell i+\ell}$  and values  $\ell, \dots, 2\ell - 1$  for the output recursion points. Pole  $p_{\ell i+1}$  cannot be a destination for a path in  $B^{(\ell)}$ , since  $\eta^U(p_{\ell i+1})$  is less than the topological order of any other pole in  $B^{(\ell)}$ . Additionally, the values of  $in$  and  $out$  need to be pairwise different or 0. Every combination of input and output vector covering the conditions formalized below in Eqs. 2–6 is valid for  $B^{(\ell)}$ . A pair  $(r_i^l, p_j) \in \mathcal{P}$  or  $(p_j, r_{i+1}^l) \in \mathcal{P}$  is a path from  $r_i^l$  to  $p_j$  or  $p_j$  to  $r_{i+1}^l$  in the set of all paths  $\mathcal{P}$  in  $B^{(\ell)}$ . Then,  $\mathcal{P}_B^{(\ell)} \subseteq \mathcal{P}$  denote the paths that are to be edge-embedded (cf. Sect. 3.1).

$$\text{INPOLEPATH:} \quad \forall l \in \{1, \dots, \ell\} : in_l \neq 0 \rightarrow (r_i^l, p_{\ell i+in_l}) \in \mathcal{P}_B^{(\ell)}, \quad (2)$$

$$\text{POLEPOLEPATH:} \quad out_l \neq 0 \wedge out_l < \ell \rightarrow (p_j, p_{\ell i+1+out_l}) \in \mathcal{P}_B^{(\ell)} \wedge \eta^U(p_j) < \eta^U(p_{\ell i+1+out_l}), \quad (3)$$

$$\text{POLEOUTPATH:} \quad out_l > \ell - 1 \rightarrow (p_{\ell i+l}, r_{i+1}^{out_l-\ell-1}) \in \mathcal{P}_B^{(\ell)}. \quad (4)$$

$$\text{INDIFF:} \quad \forall in_i, in_j \in in : i \neq j \rightarrow in_i = 0 \vee in_i \neq in_j. \quad (5)$$

$$\text{OUTDIFF:} \quad \forall out_i, out_j \in out : i \neq j \rightarrow out_i = 0 \vee out_i \neq out_j. \quad (6)$$

#### 4.2. Recursion Point Edge-Embedding

Block edge-embedding covers only the programming of the nodes within the blocks of the UC. Another task is to program the recursion points. We use the construction of [45] which, in every step, splits a  $\Gamma_2(n)$  graph in two  $\Gamma_1(n)$  graphs, which are merged to two  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graphs. This, as described later, results in a tree of graphs with fanin and fanout one or two called *supergraph* [45]. We use this supergraph for defining the paths in Valiant's 2-way EUG. For Valiant's 4-way EUG, we use every second step of the algorithm with a minor modification. We describe our modular algorithm for the 2-way and 4-way UCs below and in Listing 1.

Let  $C_{u,v}^k$  be the Boolean circuit computing function  $f$  that our UC needs to compute and  $G \in \Gamma_2(n)$  its graph representation (cf. Sect. 2.2).

1. *Splitting  $G \in \Gamma_2(n)$  in two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$* : As described in Sect. 3.1, Valiant's UC is derived from an EUG for  $\Gamma_2(n)$  graphs, which is built up of two EUGs  $(U_n^{(\ell)}(\Gamma_1))_1$  and  $(U_n^{(\ell)}(\Gamma_1))_2$  for  $\Gamma_1(n)$  graphs merged by their poles.  $G$  is similarly split into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ , which then need to be edge-embedded into  $(U_n^{(\ell)}(\Gamma_1))_1$  and  $(U_n^{(\ell)}(\Gamma_1))_2$ , respectively.  $G = (V, E) \in \Gamma_2(n)$  is split by 2-coloring its edges [45, 66], which can always be done due to König's theorem [38, 48] recapitulated in Theorems 1 and 2 on p. 7–8. After 2-coloring,

$E$  is divided into sets  $E_1$  and  $E_2$ , using which we build  $G_1 = (V, E_1)$  and  $G_2 = (V, E_2)$ , with the following conditions:

$$\text{EDGEIN}E_1\text{OR}E_2 : \forall e \in E : (e \in E_1 \vee e \in E_2) \wedge \neg(e \in E_1 \wedge e \in E_2). \quad (7)$$

$$\text{FANINI}E_1 : \forall e = (v_1, v_2) \in E_1 : \neg \exists e' = (v_3, v_4) \in E_1 : v_2 = v_4 \vee v_1 = v_3. \quad (8)$$

$$\text{FANINI}E_2 : \forall e = (v_1, v_2) \in E_2 : \neg \exists e' = (v_3, v_4) \in E_2 : v_2 = v_4 \vee v_1 = v_3. \quad (9)$$

2. *Merging a  $\Gamma_1(n)$  graph into a  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graph.* In an EUG, the number of poles decreases in each recursion step and merging a  $\Gamma_1(n)$  graph into a  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graph provides information about the paths to be taken. Let  $G_1 = (V, E) \in \Gamma_1(n)$  be a topologically ordered graph and  $G_m = (V', E') \in \Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  be a graph with nodes  $V' = \{v'_1, \dots, v'_{\lceil \frac{n}{2} \rceil}\}$ . We define two labelings  $\eta_{\text{in}}$  and  $\eta_{\text{out}}$  on  $G_m$  with  $\eta_{\text{in}}(v_i) = i$  and  $\eta_{\text{out}}(v_i) = \eta_{\text{in}}(v_i) - 1 = i - 1$ . Additionally, we define a mapping  $\theta_V$  that maps a node  $v_i \in V$  to a node  $v_j \in V'$  with  $\theta_V(v_i) = v'_{\lceil \frac{i}{2} \rceil}$ , i.e., two nodes in  $G_1$  are mapped to one node in  $G_m$ . At last, we define a mapping  $\theta_E$  that maps an edge  $e_i = (v_i, v_j) \in E$  to an edge  $e_j \in E'$  with  $\theta_E((v_i, v_j)) = (v_{\eta_{\text{in}}(\theta_V(v_i))}, v_{\eta_{\text{out}}(\theta_V(v_j))})$ , i.e., every edge in  $G_1$  is mapped to an edge in  $G_m$  as follows:  $e = (v_i, v_j) \in E$  is mapped to  $e' = (v'_k, v'_l) \in E'$ , such that  $v'_k = \theta_V(v_i)$ , and the new node of  $v_j$  in  $G_m$  is  $v'_{l+1}$  (not  $v'_l$ ).  $G_m$  is built as follows:  $V' = \{v'_1, \dots, v'_{\lceil \frac{n}{2} \rceil}\}$  and  $E' = \bigcup_{e \in E} \theta_E(e)$ . Then for all  $e = (v'_i, v'_j) \in E'$  and  $j < i$ ,  $e$  is removed from  $E'$ , along with the last node  $v'_{\lceil \frac{n}{2} \rceil}$  (due to the definition of  $\theta_E$ , it does not have any incoming edges). The resulting  $G_m$  is a topologically ordered graph in  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$ .
3. *The supergraph for Valiant's EUG construction.* In the first step,  $G$  is split into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ .  $G_1$  and  $G_2$  contain all the edges that should be embedded as paths between poles in the first and second EUGs for  $\Gamma_1(n)$ , respectively. We now explain how to edge-embed the  $\Gamma_1(n)$  graph  $G_1$  into an EUG  $U_n^{(\ell)}(\Gamma_1)$  (for  $G_2$  it is analogous).

For edge-embedding in the 2-way EUG,  $G_1$  is first merged to a  $\Gamma_2(\lceil \frac{n}{2} - 1 \rceil)$  graph  $G_m$ .  $G_m$  is then 2-colored and split into two  $\Gamma_1(\lceil \frac{n}{2} - 1 \rceil)$  graphs  $G_1^1$  and  $G_1^2$  [45]. These get merged to two graphs  $G_m^1$  and  $G_m^2$ , which are then 2-colored and split into two  $\Gamma_1(\lceil \frac{\frac{n}{2}-1}{2} - 1 \rceil)$  graphs. These steps are repeated until the recursion base is reached. In the supergraph,  $G_1^{\psi \circ 1}$  and  $G_1^{\psi \circ 2}$  are the first and second subgraphs of  $G_1^\psi$  for any  $\psi$ , respectively.

In Valiant's 4-way EUG construction [66], a supergraph that creates 4 subgraphs in each step is necessary. We require a merging method where a  $\Gamma_1(n)$  graph is merged to a  $\Gamma_4(\lceil \frac{n}{4} - 1 \rceil)$  graph where 4 nodes build a new node, and 4-color this graph to retrieve 4 subgraphs. However, this can directly be solved by using the method described above from [45]: After repeating the 2-coloring and the merging twice, we gain 4 subgraphs ( $G_1^{11}$ ,  $G_1^{12}$ ,  $G_1^{21}$  and  $G_1^{22}$ ). These can be used as if they were the result of 4-coloring the graph obtained by merging every 4 nodes into one.

However, there is a modification in this case: The first 2-coloring is a preprocessing step, which does not map to an EUG recursion step. Therefore, we have to define another

**Listing 1.** Edge-embedding algorithm for Valiant's  $\ell$ -way EUG.

---

```

1 procedure edge-embedding ( $\mathcal{U}$ ,  $G_1 = (V, E)$ )
2   Let  $\mathcal{S}$  be the set of the  $\ell$   $\Gamma_1$  subgraphs of  $G_1$  in the supergraph
3   Let  $\mathcal{R}$  be the  $\ell$  recursion step graphs
4   Let  $\mathcal{B}$  be the set of blocks in  $\mathcal{U}$ 
5   for all  $e = (v_i, v_j) \in E$  do
6     Let  $i'$  and  $j'$  denote the positions of  $v_i$  and  $v_j$  in their blocks
7      $b_i \leftarrow \lceil \frac{i}{\ell} \rceil$ ,  $b_j \leftarrow \lceil \frac{j}{\ell} \rceil$  // number of block in which  $v_i$  and  $v_j$  are
8     Let  $out$  [ $r_1$ ] denote the output vector [recursion points] of  $\mathcal{B}[b_i]$ 
9     Let  $in$  [ $r_0$ ] denote the the input vector [recursion points] of  $\mathcal{B}[b_j]$ 
10    if  $b_i = b_j$  do //  $v_i$  and  $v_j$  are in the same block
11      if  $v_i \neq v_j$  do
12         $out_{i'} \leftarrow j' - 1$ 
13      end if
14    else //  $v_i$  and  $v_j$  are in different blocks
15      Let  $s = (V', E') \in \mathcal{S}$  denote the  $\Gamma_1$  graph with  $e' = (p_{b_i}, p_{b_{j-1}}) \in E'$  and  $e'$  is not marked
16      Mark  $e'$ 
17      Let  $x$  denote the number with  $s = \mathcal{S}[x]$ 
18      Set the control bit of  $r_0^x$  to 1
19      if  $b_j = b_i + 1$  do //  $b_j$  and  $b_i$  are neighbours
20         $y \leftarrow 0$ 
21      else
22         $y \leftarrow 1$ 
23      end if
24      Set the control bit of  $r_1^x$  to  $y$ 
25       $out_{i'} \leftarrow x + \ell$ ,  $in_x \leftarrow j'$ 
26    end if
27  end for
28  Edge-embed all blocks in  $\mathcal{B}$  // edge-embed all sub-blocks
29  for  $i = 1$  to  $\ell$  do
30    if  $\mathcal{S}[i]$  exists do
31      call edge-embedding( $\mathcal{R}[i]$ ,  $\mathcal{S}[i]$ )
32    end if
33  end for
34 end procedure

```

---

labeling  $\eta_{out_P}(v) = \eta_{in}(v)$ , since in this preprocessing step we need to keep node  $v_{\lceil \frac{n}{2} \rceil}$ . Then the creation of the supergraph for the 4-way EUG construction works as follows: We merge  $G_1$  to a  $\Gamma_2(\lceil \frac{n}{2} \rceil)$  graph with labeling  $\eta_{in}$  and  $\eta_{out_P}$  and get  $G_m$ . After that, we split  $G_m$  into two  $\Gamma_1(\lceil \frac{n}{2} \rceil)$  graphs  $G_1^1$  and  $G_1^2$ . These get merged to  $\Gamma_2(\lceil \frac{n}{4} \rceil - 1)$  graphs  $G_m^1$  and  $G_m^2$  using the  $\eta_{in}$  and  $\eta_{out}$  labelings. Finally, these two graphs get split into 4  $\Gamma_1(\lceil \frac{n}{4} \rceil - 1)$  graphs  $G_1^{11}$ ,  $G_1^{12}$ ,  $G_1^{21}$ , and  $G_1^{22}$ . These are the relevant graphs for the first recursion step in Valiant's 4-way EUG construction. Then we continue for all 4 subgraphs until we reach the recursion base.

**$\ell$ -way Edge-Embedding Algorithm.** In Listing 1, we combine block edge-embedding and recursion point edge-embedding.

Let  $\mathcal{U}$  denote the part of  $U_n^{(\ell)}(\Gamma_1)$  without recursion steps (the main skeleton) and  $G_1 = (V, E)$  be the  $\Gamma_1(n)$  graph which is to be edge-embedded in  $U_n^{(\ell)}(\Gamma_1)$ .  $\mathcal{S}$  denotes the set of  $\ell$  subgraphs of  $G_1$  in the supergraph, i.e.,  $\mathcal{S} = \{G_1^1, G_1^2\}$  for  $\ell = 2$  and  $\mathcal{S} = \{G_1^{11}, G_1^{12}, G_1^{21}, G_1^{22}\}$  for  $\ell = 4$ . A *recursion step graph* of  $\mathcal{U}$  is one of the graphs having one of the  $\ell$  sets of recursion points as poles (e.g.,  $r_1^1, \dots, r_{\lceil \frac{n}{\ell} \rceil - 1}^1$ ) without the recursion steps.  $\mathcal{R}$  denotes the set of all  $\ell$  recursion step graphs of  $\mathcal{U}$ , and  $\mathcal{B}$  denotes the set of all blocks in  $\mathcal{U}$ .

We give a brief explanation of Listing 1 that describes the edge-embedding process. For any edge  $e = (v_i, v_j) \in E$  in  $G_1$ ,  $b_i$  and  $b_j$  denote the block numbers in which  $v_i$  and  $v_j$  are. We distinguish between two cases:

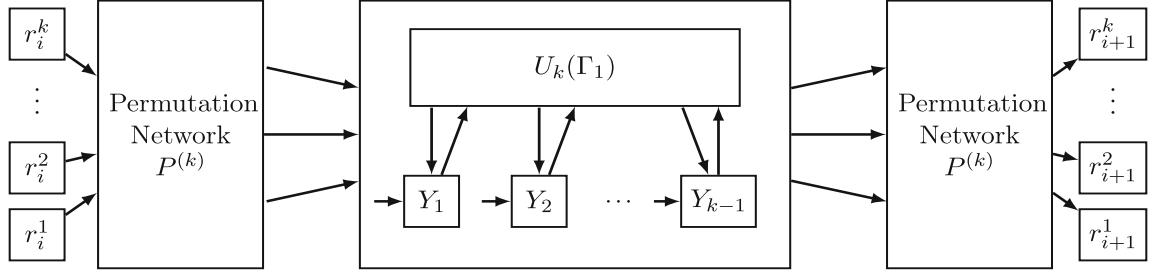
*Case 1.  $v_i$  and  $v_j$  are in the same block:  $b_i = b_j$ .* The edge-embedding is solved within the block, and no recursion points have to be programmed for the path. Therefore, vector *out* of block  $\mathcal{B}[b_i]$  is set accordingly.

*Case 2.  $v_i$  and  $v_j$  are in different blocks:  $b_i \neq b_j$ .* There exists an edge  $e' = (b_i, b_{j-1})$  in one of the  $\ell \Gamma_1(\lceil \frac{n}{\ell} - 1 \rceil)$  subgraphs of  $G_1$  that is not yet used for an edge-embedding. This determines that the path in the next recursion step has to be between poles  $p_{b_i}$  and  $p_{b_{j-1}}$ . We denote with  $s \in \mathcal{S}$  the subgraph of  $G_1$  which contains  $e'$  and  $x$  denotes its number in  $\mathcal{S}$ , i.e.,  $\mathcal{S}[x] = s$ . This implies in which of the  $\ell$  recursion step graphs we need to edge-embed the path from  $p_{b_i}$  to  $p_{b_{j-1}}$ , and so which recursion points we need to program. We first set the control bit of the  $x$ th input (resp. output) recursion points to 1 since the path between the poles with labeling  $i$  and  $j$  enters (resp. exits) the next recursion step over this recursion point. A special case to be considered here is when blocks  $\mathcal{B}[b_i]$  and  $\mathcal{B}[b_j]$  are neighbors (i.e.,  $b_j = b_i + 1$ ). Then, the path enters and leaves the next recursion step graph at the same node, whose control bit thus has to be 0. The output vector of block  $\mathcal{B}[b_i]$  is the  $i'$ th value to the  $x$ th recursion point, and the input vector of block  $\mathcal{B}[b_j]$  is the  $x$ th value to the  $j'$ th pole in this block.

We repeat these steps for all edges  $e \in E$ . Since all input and output vector of all blocks in  $\mathcal{B}$  are set, they can be embedded with the block edge-embedding. For all  $\ell$  subgraphs of  $G_1$  in the supergraph and in the EUG, we call the same procedure with  $\mathcal{S}[i] \in \mathcal{S}$ ,  $\mathcal{R}[i] \in \mathcal{R}$ ,  $1 \leq i \leq \ell$ .

## 5. Extensions to Valiant's UC Constructions

Here, we describe ideas for novel UC constructions and implementations. Firstly, in Sect. 5.1, we describe the  $k$ -way generalization of Valiant's UC presented by Lipmaa et al. in [46]. In Sect. 5.2, we describe our modular building blocks for a potentially more efficient 3-way UC. We show that Valiant's optimized  $U_3(\Gamma_1)$  cannot directly be applied as a building block in the construction due to the fact that it must have an additional node to be part of a generic EUG. We prove that the EUG without this node is not a valid EUG by showing a counterexample. Therefore, it actually results in a worse asymptotic size than Valiant's 2-way and 4-way UCs [66]. Thereafter, in Sect. 5.3, we propose a *hybrid UC*, utilizing both Valiant's 2-way and 4-way UCs or Valiant's 2-way and Zhao et al.'s 4-way UC [72] so that the overall size of the resulting hybrid UC is minimized and is at least as efficient as the better construction for the given size (in Sect. 6.2 we show its concrete improvement). Finally, in Sect. 5.4, we propose a different modular and scalable approach of Valiant's 4-way UC. This approach requires a lot of modifications in the UC generation and programming algorithm, but can be generalized to any  $k$ -way UC or to our hybrid UC.



**Fig. 6.**  $k$ -way EUG construction  $U_n^{(k)}(\Gamma_1)$  [46].

### 5.1. Generalized $k$ -way UC

In [46], Lipmaa et al. generalize Valiant's approach by providing a UC with any number of recursion points  $k$ , the so-called  $k$ -way EUG or UC. We note that their construction slightly differs from Valiant's EUG, since they do not consider the restriction on the fanout of the poles, i.e., the nodes in the EUG that correspond to universal gates or inputs (cf. Sect. 3.1). This optimization has also been included in [45] when translating an EUG to a UC, but including it in the block design leads to better sizes for the number of XOR gates. This, however, does not make a difference in case of our most prominent application of private function evaluation (PFE) (cf. Sect. 1.1), where XOR gates are free, i.e., do not require cryptographic operations and communication.

The idea is to split  $n = u + v + g$  in  $m = \lceil \frac{n}{k} \rceil$  blocks as shown in Fig. 6. Every block  $i$  consists of  $k$  inputs  $r_i^1, r_i^2, \dots, r_i^k$  and  $k$  outputs  $r_{i+1}^1, r_{i+1}^2, \dots, r_{i+1}^k$  as well as  $k$  poles, except for the last block which has a number of poles depending on  $n \bmod k$ . For every  $j \leq k$ , the list of all  $r_i^j$  builds the poles of the  $j$ th subgraph of the next recursion step, i.e., we have  $k$  subgraphs. Additionally, every block begins and ends with a Waksman permutation network [67] such that the inputs and outputs can be permuted to any pole. A Y-switching block is placed in front of every pole  $p_i$  which is connected to the  $i$ th output of the permutation network as well as the  $i$ th output of a block-intern EUG  $U_k(\Gamma_1)$ . This means that Lipmaa et al. in [46] reduce the problem of finding an efficient  $k$ -way EUG  $U_n^{(k)}(\Gamma_2)$  block  $B^{(k)}$  to the problem of finding the smallest EUG  $U_k(\Gamma_1)$ . Their solution is to build the block-intern EUG with the UC of [44], which was claimed to be more efficient for smaller circuits than [66]. Moreover, they calculate the optimal  $k$  value to be around 3.147 with their construction, which implies that the best solutions are found using small EUGs, for which Valiant provides hand-optimized solutions (i.e., for  $k = 2, 3, 4, 5, 6$ ) [66].

We note that the results recently presented by Zhao et al. [72] do not fit into this generalized  $k$ -way construction. Therefore, Zhao et al.'s optimized 4-way block is an optimization over Valiant's modular 4-way block construction [66].

#### Programming the Generalized UC

In this section, we extend the recent work of [46] by providing a detailed and modular embedding mechanism for any  $k$ -way EUG construction. We provide the main differences to the edge-embedding of the 2-way and 4-way EUG detailed in Sect. 4.



*k-way Block Edge-Embedding.* In this setting, our main block is a programmable block  $B^{(k)}$  with  $k$  poles  $p_1, \dots, p_k$ , and  $k$  input [output] recursion points  $r_0^1, \dots, r_0^k$  [ $r_1^1, \dots, r_1^k$ ].  $B^{(k)}$  is topologically ordered with mapping  $\eta^U$  as defined in Sect. 2.1. Vectors  $in = (in_1, \dots, in_k) \in \{0, \dots, k\}^k$  and  $out = (out_1, \dots, out_k) \in \{0, \dots, 2k-1\}^k$  denote the input and output vectors of  $B^{(k)}$ , respectively. Values  $k, \dots, 2k-1$  in  $out$  denote the recursion point targets  $r_1^1, \dots, r_1^k$  (cf. Sect. 4.1). The setting of  $in$  and  $out$  is formalized in Eqs. 2–6 when  $\ell = k$ .

*k-way Recursion Point Edge-Embedding.*  $G \in \Gamma_2(n)$  denotes the transformed graph of a Boolean circuit  $C_{u,v}^g$ , where  $n = u + v + g$ .

1. *Splitting*  $G \in \Gamma_2(n)$  into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$ : Similarly as in Sect. 4.2, we first split  $G$  into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$  with 2-coloring.
2. *Merging a  $\Gamma_1(n)$  graph into a  $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$  graph*  $G_1 = (V, E) \in \Gamma_1(n)$  is merged into a  $\Gamma_k(\lceil \frac{n}{k} - 1 \rceil)$  graph  $G_m = (V', E')$  (same for  $G_2$ ). Therefore, we redefine mapping  $\theta_V$  (cf. Sect. 4.2) that maps node  $v_i \in V$  to node  $v_j \in V'$ . In this scenario,  $k$  nodes in  $V$  build one node in  $V'$ , so  $\theta_V(v_i) = v_{\lceil \frac{i}{k} \rceil}$ . The mapping of the edges  $\theta_E$  is the same as in the 2-way and 4-way EUG construction, and  $(v'_i, v'_j) \in E'$  where  $j < i$  edges are removed along with  $v_{\lceil \frac{i}{k} \rceil}$  in the end.  $G_m$  is then a topologically ordered graph in  $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$ .
3. *The supergraph for Lipmaa et al.'s  $k$ -way EUG construction* The next step of the construction is to split  $G_m \in \Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$  into  $k$   $\Gamma_1(\lceil \frac{n}{k} - 1 \rceil)$  graphs. This is done with  $k$ -coloring: A directed graph  $K = (V, E)$  can be  $k$ -colored, if  $k$  sets  $E_1, \dots, E_k \subseteq E$  cover the following conditions:

$$\text{DISJOINT } \forall i, j \in \{1, \dots, k\} : i \neq j \rightarrow E_i \cap E_j = \emptyset. \quad (10)$$

$$\text{EDGEIN } E_i \quad \forall e \in E : \exists i \in \{1, \dots, k\} : e \in E_i. \quad (11)$$

$$\begin{aligned} \text{FANIN } E_i \quad \forall i \in \{1, \dots, k\}, \forall e = (v_1, v_2) \in E_i : \\ \neg \exists e' = (v_3, v_4) \in E_i \setminus \{e\} : v_2 = v_4 \vee v_1 = v_3. \end{aligned} \quad (12)$$

According to Kőnig's theorem [38, 48] described in Sect. 2.1,  $\Gamma_k(n)$  graphs can always be  $k$ -colored efficiently with a dedicated algorithm. The rest of the supergraph construction and the way it is used for edge-embedding is the same as for the 2-way and 4-way EUG as described in Sect. 4.2.

*k-way Edge-Embedding Algorithm.* The edge-embedding algorithm is the same as shown in Listing 1, with  $\ell = k$ .

## 5.2. Potentially More Efficient 3-Way UC

The optimal  $k$  value for minimizing the size of the  $k$ -way UC was calculated to be 3.147 in [46]. We describe our idea of a 3-way UC. Intuitively, based on an optimization by Valiant [66], this UC should result in the best asymptotic size. The asymptotic size of any  $k$ -way UC depends on the size of its modular body block  $B^{(k)}$  (e.g., Fig. 5a or b on

p. 14 for the 4-way UC). Once it is determined, the size of the UC is  $\text{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \text{size}(U_n^{(k)}(\Gamma_1)) \sim 2 \cdot \frac{\text{size}(B^{(k)})}{k} n \log_k n = 2 \cdot \frac{\text{size}(B^{(k)})}{k \log_2(k)} n \log_2 n$ . The modular block consists of two permutation networks  $P^{(k)}$ , an EUG  $U_k(\Gamma_1)$ , and  $(k - 1)$  Y-switching blocks (cf. Sect. 5.1, [46]).<sup>2</sup>

**Size of Body Block  $B^{(3)}$  with Valiant's Optimized  $U_3(\Gamma_1)$ .** According to Valiant [66], an EUG  $U_3(\Gamma_1)$  with 3 poles contains only three-connected poles (used as recursion base in Sect. 3.1). An optimal permutation network  $P^{(3)}$  that achieves the lower bound has 3 nodes as well. This implies that  $\text{size}(B^{(k)}) = 2 \cdot P^{(3)} + \text{size}(U_3(\Gamma_1)) + (3 - 1) = 11$ . Then, the size of the UC becomes  $\sim 2 \cdot \frac{11}{3 \log_2 3} n \log_2 n \sim 4.627 n \log_2 n$ , which means an asymptotically by around 2.5% smaller size than that of Valiant's 4-way UC with  $\sim 4.75 n \log_2 n$ .

However, there is a flaw in this initial design. Valiant's  $U_3(\Gamma_1)$  only works as an EUG for 3 nodes under special conditions, e.g., when it is a subgraph within a larger EUG. There are 3 possible edges in a topologically ordered graph  $G = (V, E)$  in  $\Gamma_1(3)$ :  $(1, 2)$ ,  $(2, 3)$  and  $(1, 3)$ .  $(1, 2)$  and  $(2, 3)$  can be directly embedded in  $U_3(\Gamma_1)$  using  $(p_1, p_2)$  and  $(p_2, p_3)$ , respectively.  $(1, 3)$ , however, has to be embedded as a path *through* node 2, i.e., as a path  $((p_1, p_2), (p_2, p_3))$ . When  $U_3(\Gamma_1)$  is a subgraph of a bigger EUG, this is possible by programming  $p_2$  accordingly. However, when we use this  $U_3(\Gamma_1)$  as a building block in the body block of our EUG, it cannot directly be applied, due to the fact that the programming of  $p_2$  depends on other constraints as well. A generic  $U_3(\Gamma_1)$  that can embed  $(1, 3)$  without going through  $p_2$  as before has an additional Y-switching block between  $p_2$  and  $p_3$ .

We depict in Fig. 7a the 3-way body block that uses Valiant's optimized  $U_3(\Gamma_1)$  in the  $k$ -way block design of [46] and show that it is not a valid body block for an EUG construction. Assume that the output of pole  $p_{3i+1}$  has to be directed to pole  $p_{3i+3}$  (green path). Then, it needs to go through pole  $p_{3i+2}$ , which means that the red edge going to  $p_{3i+2}$  is used by this path. However, there can be an other edge coming from the permutation network as an input to  $p_{3i+2}$ , e.g., from  $p_{3i}$  from the preceding block through  $r_i^1$  (blue path). This cannot be directed to  $p_{3i+2}$  anymore, as shown in Fig. 7a, since the red edge would carry two different values. Therefore, in the 3-way body block construction, it does not suffice to use Valiant's optimized  $U_3(\Gamma_1)$  [66].

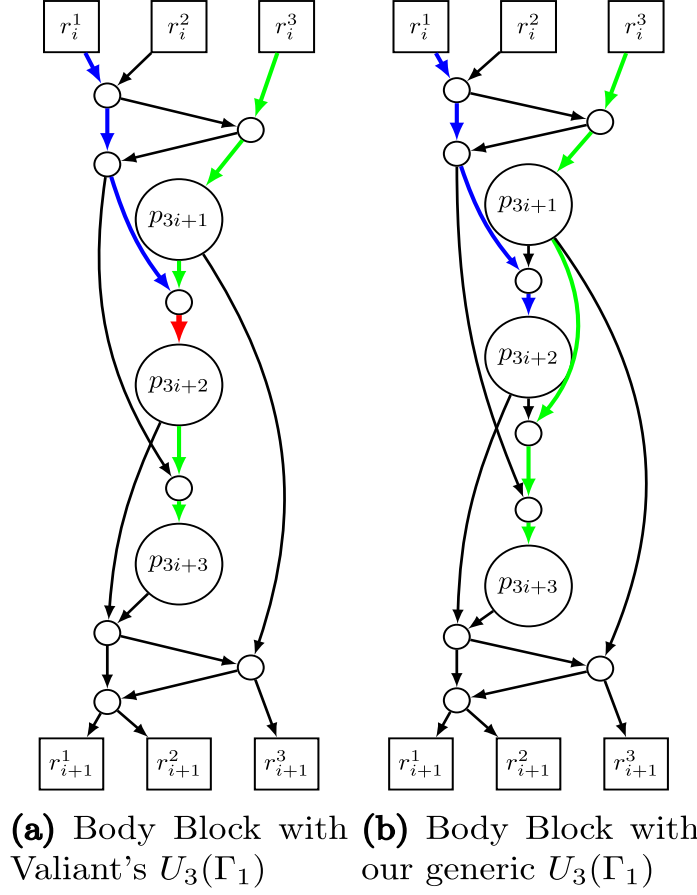
**Size of Body Block  $B^{(3)}$  with Our Generic  $U_3(\Gamma_1)$ .** In Fig. 7b, we show the 3-way body block with the generic  $U_3(\Gamma_1)$  that allows the output from  $p_{3i+1}$  to be directed to  $p_{3i+3}$  without having to go *through*  $p_{3i+2}$  (green path), and the edge going into  $p_{3i+2}$  can be utilized by the path directed into this node (blue path). This results in  $\text{size}(B^{(3)}) = 2 \cdot P^{(3)} + \text{size}(U_3(\Gamma_1)) + (3 - 1) = 12$ , which implies that the size of the UC is  $\sim 2 \cdot \frac{12}{3 \log_2 3} n \log_2 n = 5.047 n \log_2 n$ . Unfortunately, this is even worse than the size of the 2-way UC with  $\sim 5 n \log_2 n$ , and we therefore conclude that the most efficient known UC is Valiant's 4-way UC with Zhao et al.'s optimization.

Recently, Zhao et al. [72] have shown by exhaustive search over all possible topologies that the 3-way body block  $B^{(3)}$  presented in Fig. 7b results in the smallest 3-way UC by

---

<sup>2</sup>We note that in this section, we design the body block according to [46], i.e., the poles do not have a fanout restriction. However, all other nodes have fanout-2 restriction.





**Fig. 7.** Body block  $B^{(3)}$  construction for our 3-way EUG  $U_n^{(3)}(\Gamma_1)$ .

showing that no block with only 11 additional nodes can be used as a universal block, and indeed, our block with 12 additional nodes can be utilized.

### 5.3. 2/4 Hybrid UC Construction

In this section, we detail our hybrid UC based on Valiant's 2-way and 4-way UCs with the optimization by Zhao et al. [72], which yields the smallest UCs to date. Given the size of the input circuit  $C_{u,v}^g$ , i.e.,  $n = u + v + g$ , we can calculate at each recursion step if it is better to create 2 subgraphs of size  $\lceil \frac{n}{2} - 1 \rceil$  and utilize the 2-way recursive skeleton, or it is more beneficial to create a 4-way recursive skeleton with 4 subgraphs of size  $\lceil \frac{n}{4} - 1 \rceil$ .

We assume that for every  $n$ , we have an algorithm that computes the size (i.e.,  $\text{size}(U_n^{\text{hybrid}(K)}(\Gamma_1))$ ) of the hybrid UC for sizes smaller than  $n$ . We give details on how it is computed in Sect. 6. Then, Listing 2 describes the algorithm for constructing a hybrid UC, at each step based on which strategy is more efficient. We note that our hybrid construction is generic, and given multiple  $k$ -way UCs as parameter  $K$  ( $K = \{2, 4\}$  in our example), it minimizes the concrete size of the resulting UC.

**Listing 2.** Hybrid construction algorithm, where  $B^{(k)}(i)$ ,  $H^{(k)}(i)$  and  $T^{(k)}(i)$  denote body, head and tail blocks with  $i$  poles in the  $k$ -way UC, respectively.

---

```

1 procedure hybrid ( $p_1, \dots, p_n, K = \{2, 4\}$ )
2   Let  $\text{size}(U_{n'}^{\text{hybrid}(K)}(\Gamma_1))$  be the function calculating the size of the smaller hybrid constructions with
    $\hookrightarrow \text{size } n' \leq n$ 
3   for all  $k \in K$  do // Number of poles in the last block for all  $k$ 
4     if  $n \mid k$  do
5        $m_k \leftarrow k$ 
6     else
7        $m_k \leftarrow n \bmod k$ 
8     end if
9      $s_k \leftarrow \text{size}(H^{(k)}(k)) + \left(\lceil \frac{n}{k} \rceil - 3\right) \cdot \text{size}(B^{(k)}(k)) + \text{size}(B^{(k)}(r_k)) + \text{size}(T^{(k)}(m_k)) +$ 
        $\hookrightarrow m_2 \cdot \text{size}\left(\text{size}(U_{\lceil \frac{n}{2} \rceil}^{\text{hybrid}(K)}(\Gamma_1))\right) + ((k - m_k) \cdot \text{size}\left(\text{size}(U_{\lfloor \frac{n}{k} \rfloor}^{\text{hybrid}(K)}(\Gamma_1))\right))$ 
10  end for
11   $s_i \leftarrow \min(s_k : k \in K)$  // Choose the better construction
12  // GENERATION
13  Create skeleton for  $i$ -way construction with  $n$  poles
14  call hybrid( $r_1^1, \dots, r_{\lceil \frac{n}{i} \rceil}^1, K$ ), ..., hybrid( $r_1^{m_i}, \dots, r_{\lceil \frac{n}{i} \rceil}^{m_i}, K$ )
15  if  $(i - m_i) > 0$  do call hybrid( $r_1^{m_i}, \dots, r_{\lfloor \frac{n}{i} \rfloor}^{m_i}, K$ ), ..., hybrid( $r_1^i, \dots, r_{\lfloor \frac{n}{i} \rfloor}^i, K$ )
16  end if
17  // PROGRAMMING
18  Call edge-embedding( $\mathcal{U}^{(i)}, G_1^{(i)} = (V, E)$ ) // Call embedding algorithm corresponding to  $i$ 
19 end procedure

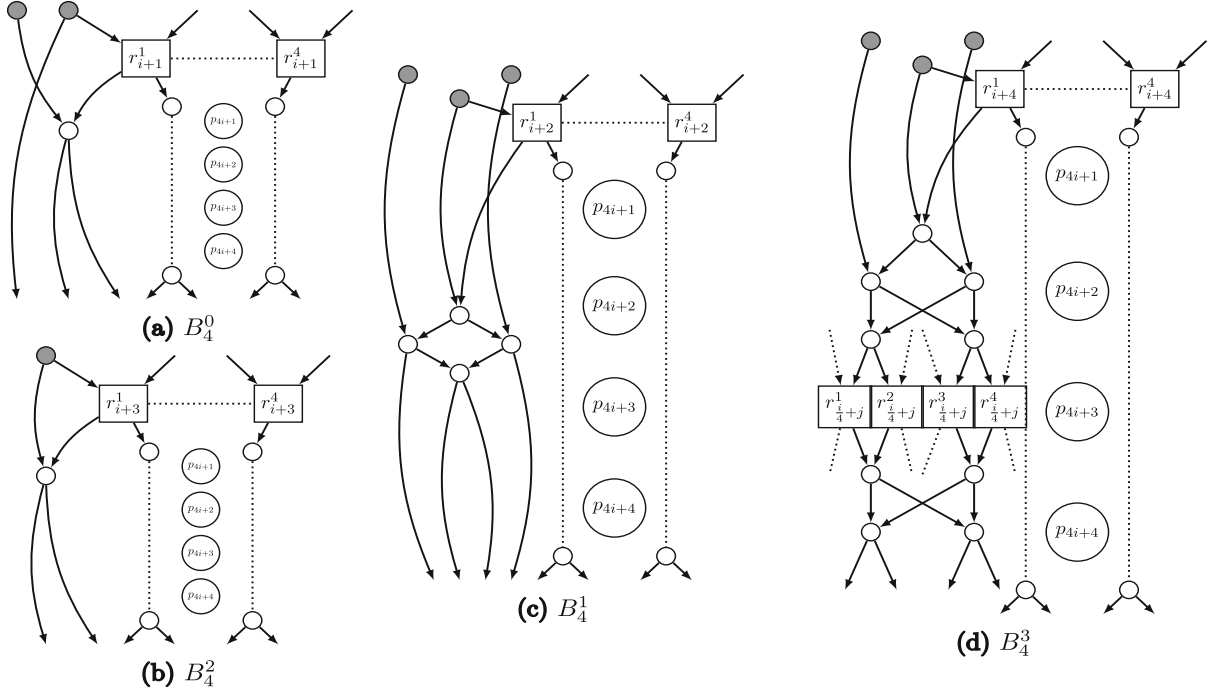
```

---

#### 5.4. Scalable 4-way UC Construction

Our existing implementations of [31, 45] store the whole UC of size  $\mathcal{O}(n \log n)$  in memory, which therefore becomes a bottleneck when it comes to scalability. In this section, we present the design of our scalable universal circuit construction. Specifically, we show how Valiant's 4-way UC can be modified to use  $\mathcal{O}(n)$  memory in the input circuit size  $n$  at each step of the execution. We note that our approach is generic, and with additional implementation effort, it can be extended to any  $k$ -way UC as well as for the 4-way UC of Zhao et al. [72].

In this section, we present our design that utilizes two separate phases. The first phase is *scalable UC generation* (Sect. 5.4.1), where the universal circuit is generated given the size  $n$  of the input circuit. This is solved by generating the topologically ordered UC layer by layer, each of which has size  $\mathcal{O}(n)$ . The output of this step is a set of circuit files, which all contain a subgraph of size  $\mathcal{O}(n)$ , which helps to significantly reduce the complexity of the second phase, i.e., *scalable UC programming* (Sect. 5.4.2). In this step, the subcircuits resulting from the first phase are programmed individually, i.e., we proceed subcircuit by subcircuit instead of edge by edge of the input circuit as before. Therefore, the output of this step is a set of programming files that contain the programming bits respective to the circuit files. In Sect. 7.2, we will show experimentally that our scalable UC construction significantly reduces the memory usage.



**Fig. 8.** Scalable body block construction. **a** The first part  $B^0 = B_4^0$  of the body block, **c** the second  $B^1 = B_4^1$ , **b** the third  $B^2 = B_4^2$ , and **d** the fourth  $B^3 = B_4^3$ , where further subgraphs are created. We note that the nodes are shown only for one of the four subgraphs, but they are the same for all four subgraphs. Scalable head and tail blocks are designed analogously.

#### 5.4.1. Scalable Per-Block UC Generation

The underlying idea behind our scalable UC generation is to generate the blocks of the main skeleton one by one, only keeping one such block and its corresponding subgraph nodes in memory at once. In this scenario, these blocks will be regarded as layers. Additionally, we store some necessary information from the preceding three layers in dedicated files, but delete these as soon as they become redundant. The required additional information is the topological order of nodes that are already defined and have edges directed into the current layer. Since the number of subgraphs in any layer is  $\mathcal{O}(n)$ , the number of nodes held in memory at any point is  $\mathcal{O}(n)$  as well, since in each layer there are only a constant number of nodes.

Our scalable UC generation relies on the fact that at each block of the main skeleton, based on the modulo 4 result for each next recursion step, we know which part of the next subgraph skeleton or potentially recursion base graph we build at each layer. This observation helps us reconstruct how the subgraphs may look like for a given body block in Valiant's 4-way UC. Since the structure of this is complicated and there are many cases to consider, we show in Fig. 8 the cases for Valiant's body block from Fig. 5a on p. 14 [66] and note that head and tail blocks can be constructed analogously. Moreover, a similar scalable design can be constructed for Zhao et al.'s body block (Fig. 5b) [72].

Figure 8d shows a recursive block construction with Figs. 8b, c being base cases. From Fig. 8, each body block construction type is denoted by  $B^i$  where  $i = \{0, 1, 2, 3\}$ <sup>3</sup> is the

<sup>3</sup>Note that our design corresponds to Valiant's 4-way UC, but for simplicity, we use  $B^i$  instead of  $B^{(4)i}$ .

**Table 1.** Files storing the UC in our scalable UC generation for an example with  $n = 36$ .

$f^{44}$	$f^{43}$	$f^{42}$	$f^{41}$	$f^4$	...	$f^{14}$	$f^{13}$	$f^{12}$	$f^{11}$	$f^1$	$f^0$	$g^1$	$g^{11}$	$g^{12}$	$g^{13}$	$g^{14}$	...	$g^4$	$g^{41}$	$g^{42}$	$g^{43}$	$g^{44}$
$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$	$H^0$	...	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$	$H^0$	$B$	$H^0$	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$	...	$H^0$	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$
				$H^1$	...					$H^1$	$B$	$H^1$					...	$H^1$				
				$H^2$	...					$H^2$	$B$	$H^2$					...	$H^2$				
				$H^3$	...	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$	$H^3$	$B$	$H^3$	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$	...	$H^3$	$R_1^0$	$R_1^0$	$R_1^0$	$R_1^0$
				$T_4^0$	...					$T_4^0$	$B$	$T_4^0$					...	$T_4^0$				
				$T_4^1$	...					$T_4^1$	$B$	$T_4^1$					...	$T_4^1$				
				$T_4^2$	...					$T_4^2$	$B$	$T_4^2$					...	$T_4^2$				
				$T_4^3$	...					$T_4^3$	$T$	$T_4^3$					...	$T_4^3$				
1	1	1	1	8	...	1	1	1	1	8	36	8	1	1	1	1	...	8	1	1	1	1

position of nodes between two poles in a body block in the subgraph. A given subgraph has node(s) between every two set of recursion points of the parent graph to which this subgraph belongs. We know that the recursion points, for instance  $\{r_1^1, \dots, r_{\lceil \frac{n-4}{4} \rceil}^1\}$ , are the poles of the next recursion step subgraph. Analogously, we can design head  $H^i$ , tail  $T_x^i$ , and special last body blocks  $B_x^i$ , where  $x = \{1, 2, 3, 4\}$  denotes the type of the body or tail block based on the number of input or output recursion points, respectively. In the following, we use an example to detail how our scalable UC generation works. We depict the resulting UC files and what their content is in Table 1.

*Generation of first (main) skeleton.* Generating the first (main) skeleton of the two  $U_n(\Gamma_1)$  EUGs that are merged into a  $U_n(\Gamma_2)$ , EUG differs from the next, recursive steps. Let us consider an example of a DAG with  $n = u + k + v = 36$ . Ideally, our approach constructs twice the same block from the left and right  $U_n(\Gamma_1)$  EUGs. In this scenario for  $U_n(\Gamma_1)$ , we have one (merged) head block  $H$ , seven (merged) body blocks  $B$ , and one (merged) tail block  $T_4$  with 4 nodes in the main skeleton. Constructing the first head block is straightforward according to [31, Fig. 4e] as we do not have to construct any subgraph. Thereafter, we construct seven body blocks according to Fig. 5a and a tail block according to [31, Fig. 4f]. However, these merged blocks require constructing the subgraph nodes in the same layer alongside with it, as we describe next. Note that in this first step, we actually generate twice the four sets of subgraph nodes, since the two  $U_n(\Gamma_1)$  EUGs are merged into a  $U_n(\Gamma_2)$  EUG (cf. Sect. 3.1), but in later recursion steps, only four sets of subgraph nodes are generated.

*Generating subgraph nodes recursively per layer.* We can generate the subgraph nodes recursively for all recursion steps at a given position for nodes  $n$ . In our example with  $n = 36$ , we only have a head and a tail block for the recursion graph with  $\lceil \frac{n-4}{4} \rceil = 8$  poles. Therefore, we construct the first body block with  $H^0$  as subgraph level, the second body block with  $H^1$ , thereafter  $H^2$  and  $H^3$ . The fifth body block is constructed with  $T^0$ , the sixth and seventh with  $T^1$  and  $T^2$ , respectively, and the tail block with  $T^3$ . Recursive scalable blocks are  $H^3$  and  $B^3$  as shown in Fig. 8d.  $T_4^3$  does not have recursion points anymore, since a tail block has no output recursion points. For  $n = 8$ , we reach a

recursion base with  $\lceil \frac{n-4}{4} \rceil = 1$ . However, for a larger  $n$ , more recursion steps might be necessary. Therefore, at each layer, we generate all subgraph nodes necessary, and if a recursion step, i.e.,  $H^3$  or  $B^3$ , occurs, we generate the nodes of the next subgraph as well, etc. We denote the recursion bases by  $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$  with 1, 2, 3, and 4 nodes, respectively.

With this, we have shown how to generate topologically ordered universal circuits using the file system and achieve a scalable algorithm for UC generation that stores at most  $\mathcal{O}(n)$  information in memory. Moreover, our approach requires  $4.75n \log_2 n$  disk space to store the universal circuit as before, and additionally  $\mathcal{O}(n)$  extra storage space for every layer. However, we only store additional data for the prior three layers and delete any other stored data at each step. In the end of the UC generation, we can delete any additionally stored data. The maximum storage requirement for our algorithm is before deleting the additionally stored data for the last layer, since the size of the UC dominates the storage requirements at any other step (when only a part of it is generated yet).

#### 5.4.2. Scalable UC Programming

As described in Sect. 5.4.1, we design our scalable UC generation such that each subgraph is written into a separate file. This is important to also allow the programming step to require only  $\mathcal{O}(n)$  memory. It can be observed in Listing 1 on p. 17 that the recursion point edge-embedding algorithm inherently handles the UC subgraph by subgraph (cf. Sect. 4.2), which in turn calls the block edge-embedding for all blocks in a subgraph. We observe that each skeleton can be programmed based on the information stored only in the corresponding  $\Gamma_1$  graph, and therefore, we can store the programming bits in a separate file for each subgraph in the same order as the nodes of the subgraph.

After reading a subgraph from its file resulting from the UC generation step detailed in Sect. 5.4.1, it is programmed as described in Listing 1. The embedding starts from the main skeleton in file  $f^0$  and continues with  $f^1, \dots, f^4$  and  $g^1, \dots, g^4$ , etc., and results in the corresponding programming files  $p^0, p^1, \dots, p^4$  and  $q^1, \dots, q^4$ , etc.

## 6. Size and Depth of UCs

In this section, we review the size and depth of the UCs considered in this article. The *size of the edge-universal graph*  $U_n^{(k)}(\Gamma_1)$  is the number of nodes, counting all the poles and nodes created using Valiant's construction from Sect. 3.1. The *depth of the edge-universal graph* is the number of nodes on the longest path between any two nodes, i.e., essentially the path between the first input and last output.  $U_n^{(k)}(\Gamma_2)$  is built from two  $U_n^{(k)}(\Gamma_1)$  edge-universal graphs as described in Sect. 3.1. When transforming  $U_n^{(k)}(\Gamma_2)$  into a UC, the first  $u$  poles are associated with inputs, the last  $v$  poles with outputs, and the  $g$  poles between are realized with universal gates (cf. Eq. 1 on p. 11) whose programming is defined by the corresponding gates in the simulated circuit. The rest of the nodes of  $U_n^{(k)}(\Gamma_2)$  are translated into universal programmable (X and Y) switching blocks (cf. Fig. 2 on p. 11), whose programming is defined by the edge-embedding of

the graph of the circuit  $G$  into  $U_n^{(k)}(\Gamma_2)$ . Thus, when considering the *sizes and depths of the UCs*, we realize the nodes and poles as circuit building blocks and express the concrete and asymptotic sizes in the number of switches ( $X$  and  $Y$ ) and universal gates ( $U$ ) (cf. Sect. 3.2).

In Sect. 6.1, we recapitate the asymptotic size and depth of Valiant's 2-way and 4-way UCs [66], i.e.,  $UC^{\text{Valiant-2}}$  and  $UC^{\text{Valiant-4}}$ , respectively, of Zhao et al.'s 4-way UC  $UC^{\text{Zhao et al.-4}}$  [72] and of the smallest  $k$ -way UCs following Lipmaa et al.'s generalization [46]. Thereafter, in Sect. 6.2, we present optimizations that reduce the size (and potentially the depth as well) of UCs, regardless of which constructions were used for their generation. We revise the concrete sizes and depths of  $UC^{\text{Valiant-2}}$  and  $UC^{\text{Valiant-4}}$ ,  $UC^{\text{Zhao et al.-4}}$  as well as that of our 2/4 hybrid UCs  $UC^{\text{H(Valiant-2,4)}}$  and  $UC^{\text{H(Valiant-2, Zhao et al.-4)}}$  (cf. Sect. 5.3).

### 6.1. Asymptotic Size and Depth of $k$ -Way UCs

Lipmaa et al.'s  $k$ -way UC [46] is discussed briefly in Sect. 5.1 and is depicted in Fig. 6 on p. 19. They show that a  $k$ -way body block may consist of two permutation networks  $P^{(k)}$ , an EUG for  $k$  nodes, i.e.,  $U_k(\Gamma_1)$ , and additionally,  $(k - 1)$  Y-switching blocks. In this section, we recapitulate the sizes in Table 2 and depths in Table 3 of these building blocks and give an estimate for the leading constant for Lipmaa et al.'s  $k$ -way EUGs and UCs with size  $\mathcal{O}(n \log_2 n)$  and depth  $\mathcal{O}(n)$ , for  $k \in \{2, \dots, 8\}$ . We conclude that among all UCs following this generalization, the best size is achieved by Valiant's 4-way UC,  $UC^{\text{Valiant-4}}$ . This does not exclude the possibility for a more efficient UC, as has been shown in [72], where Zhao et al. propose a 4-way UC,  $UC^{\text{Zhao et al.-4}}$ , using a smaller body block. Therefore, their construction achieves the smallest asymptotic size to date. However, Zhao et al. state that their method cannot be used yet to find more efficient UCs for  $k > 4$ , since it includes an exhaustive search for which the domain becomes too large.

#### 6.1.1. Edge-Universal Graph with $k$ Poles

*Size.* Valiant optimized EUGs up to size 6 by hand in [66]: For  $k = 2$ ,  $U_2(\Gamma_1)$  has two poles, for  $k = 3$  we discussed in Sect. 5.2 that an additional node is necessary. For  $k \in \{4, 5, 6\}$ , the sizes are  $\{6, 10, 13\}$ , as shown in [45, Fig. 1] (the nodes denoted as empty circles disappear in the UC). For  $k = 7$  and  $k = 8$ , we observe that  $UC^{\text{Valiant-2}}$  results in a better size than that of  $UC^{\text{Valiant-4}}$  due to the smaller permutation network and less recursion nodes. Therefore, we use these constructions to compute the size of  $U_7(\Gamma_1)$  and  $U_8(\Gamma_1)$ . As mentioned in [46], another possibility is to use the UC of [44] instead of these EUGs since they have better sizes for small circuits. These UCs  $U_k^{\text{KS08}}$  are built from two smaller  $U_{\frac{k}{2}}^{\text{KS08}}$ , a  $P^{(\frac{k}{2})}$  and  $\frac{k}{2}$  Y switches [44]. It results in a smaller size of 21 for  $k = 8$ .

*Depth.* The depth of the hand-optimized EUGs for  $k \in \{2, 3, 4, 5, 6\}$  is, respectively,  $\{2, 4, 5, 7, 10\}$  as shown in [45, Fig. 1]. The depth of  $U_7(\Gamma_1)$  and  $U_8(\Gamma_1)$  becomes, respectively, 16 and 19 with Valiant's 2-way UC, and 14 and 16 with the UC from [44].



**Table 2.** Leading term of the asymptotic  $\mathcal{O}(n \log_2 n)$  **sizes** of  $k$ -way edge-universal graphs ( $U_n^{(k)}(\Gamma_1)$ ) and universal circuits (UC) and the concrete size of their building blocks for  $k \in \{2, \dots, 8\}$  according to the design of [46].

$k$	Reference	$U_k(\Gamma_1)$ #nodes	$U_k(\text{KS08})$ #nodes	$P_1^{(k)}$ #nodes	$P_W^{(k)}$ #nodes	$B^{(k)}$ #nodes	$U_n^{(k)}(\Gamma_1)$ #nodes ( $\cdot n \log_2 n$ )	UC #switches ( $\cdot n \log_2 n$ )
2	[66]	<b>2</b>	2	1	<b>1</b>	5	= 2.500	= 5.000
3	[31]	<b>4</b>	6	3	<b>3</b>	12	$\approx 2.524$	$\approx 5.047$
4	[66]	<b>6</b>	7	5	<b>5</b>	19	= 2.375	= 4.750
5	[46]	<b>10</b>	11	7	<b>8</b>	30	$\approx 2.584$	$\approx 5.168$
6	[46]	<b>13</b>	14	10	<b>11</b>	40	$\approx 2.579$	$\approx 5.158$
7	[46]	<b>19</b>	19	13	<b>14</b>	53	$\approx 2.697$	$\approx 5.394$
8	[46]	23	<b>21</b>	16	<b>17</b>	62	$\approx 2.583$	$\approx 5.167$
4*	[72]	–				18	= 2.250	= 4.500

4\* [72] denotes the 4-way construction with the optimized block of [72], i.e.,  $\text{UC}^{\text{Zhao et al.}-4}$ .  $n$  denotes the size of the input  $\Gamma_2(n)$  circuit,  $U_k(\Gamma_1)$  Valiant’s edge-universal graph with  $k$  poles,  $U_k^{\text{KS08}}$  the UC of [44],  $P_1^{(k)}$  the permutation network for  $k$  nodes achieving the lower bound for the size, and  $P_W^{(k)}$  Waksman’s permutation network [67].  $B^{(k)}$  is the  $k$ -way body block with the best existing alternative for universal circuits and permutation networks marked in bold

### 6.1.2. Permutation Networks $P^{(k)}$

*Size.* Waksman in [67] showed that the lower bound for the size of a permutation network is  $\lceil \log_2(k!) \rceil$  for  $k$  elements. We show this lower bound in Table 2 as  $P_1^{(k)}$ . The size of the smallest existing permutation network is Waksman’s permutation network  $P_W^{(k)}$  [7, 67]. For  $k \in \{2, 3, 4\}$ , its size matches the lower bound, but for larger values of  $k$ ,  $P_W^{(k)}$  uses additional nodes.

*Depth.* The depth of a permutation network has lower bound  $\lceil \log_2(k!) \rceil + 1$ , since each input has to have a path to each output, where switches have only two inputs and two outputs. We show these as the depth of  $P_1^{(k)}$  in Table 3. Waksman’s permutation network matches the lower bound when  $k \in \{2, 3, 4\}$ , but utilizes additional nodes for larger values of  $k$ .

### 6.1.3. Body Blocks

A body block  $B^{(k)}$  is built of  $(k - 1)$  Y-switching blocks, an EUG for  $k$  nodes, and two permutation networks  $P^{(k)}$  [46] (cf. Fig. 6 on p. 19).  $B^{(k)}$  shown in Tables 2 and 3 is built using Waksman’s permutation network  $P_W^{(k)}$ .

*Size.* The size of the body block is the sum of the sizes of its building blocks, i.e.,  $\text{size}(B^{(k)}) = \min(\text{size}(U_k(\Gamma_1)), \text{size}(U_k^{\text{KS08}})) + 2 \cdot \text{size}(P^{(k)}) + (k - 1) \cdot \text{size}(Y)$ .

*Depth.* The depth of  $B^{(k)}$  is the number of edges in its building blocks, the additional edges between the different blocks and the recursion nodes. This means that in total  $\text{depth}(B^{(k)}) = \min(\text{depth}(U_k(\Gamma_1)), \text{depth}(U_k^{\text{KS08}})) + 2 \cdot \text{depth}(P^{(k)}) + (k - 1) \cdot \text{depth}(Y) + 1$ .



**Table 3.** Leading terms of the asymptotic  $\mathcal{O}(n)$  **depths** of  $k$ -way edge-universal graphs ( $U_n^{(k)}(\Gamma_1)$ ) and universal circuits (UC) and the concrete depth of their building blocks for  $k \in \{2, \dots, 8\}$  according to the design of [46].

$k$	Reference	$U_k(\Gamma_1)$ #nodes	$U_k(\text{KS08})$ #nodes	$P_1^{(k)}$ #nodes	$P_W^{(k)}$ #nodes	$B^{(k)}$ #nodes	$U_n^{(k)}(\Gamma_1)$ #nodes ( $\cdot n$ )	UC #switches ( $\cdot n$ )
2	[66]	<b>2</b>	2	1	<b>1</b>	6	= 3.000	= 3.000
3	[31]	<b>4</b>	5	3	<b>3</b>	13	$\sim 4.333$	$\sim 4.333$
4	[66]	<b>5</b>	6	3	<b>3</b>	15	= 3.750	= 3.750
5	[46]	<b>7</b>	9	4	<b>5</b>	22	= 4.400	= 4.400
6	[46]	<b>10</b>	12	4	<b>5</b>	26	$\sim 4.333$	$\sim 4.333$
7	[46]	16	<b>14</b>	4	<b>5</b>	31	$\sim 4.429$	$\sim 4.429$
8	[46]	19	<b>16</b>	4	<b>5</b>	34	= 4.250	= 4.250
4*	[72]	–				14	= 3.500	= 3.500

4\* [72] denotes the 4-way construction with the optimized block of [72], i.e.,  $\text{UC}^{\text{Zhao et al.-4}}$ .  $n$  denotes the size of the input  $\Gamma_2(n)$  circuit,  $U_k(\Gamma_1)$  Valiant’s edge-universal graph with  $k$  poles,  $U_k^{\text{KS08}}$  the UC of [44],  $P_1^{(k)}$  the permutation network for  $k$  nodes achieving the lower bound for the depth, and  $P_W^{(k)}$  Waksman’s permutation network [67].  $B^{(k)}$  is the  $k$ -way body block with the best existing alternative for universal circuits and permutation networks marked in bold

#### 6.1.4. Edge-Universal Graphs and Universal Circuits with $n$ Poles

Two  $k$ -way EUGs  $U_n^{(k)}(\Gamma_1)$  graphs build up an EUG  $U_n^{(k)}(\Gamma_2)$  as described in Sect. 3.1. *Size.* The asymptotic size of EUG  $U_n^{(k)}(\Gamma_1)$  is determined as  $\text{size}(U_n^{(k)}(\Gamma_1)) = \frac{\text{size}(B^{(k)})}{k \log_2 k} n \log_2 n$ . The leading factor for a  $\text{size}(UC)$  is twice this number, since asymptotically, the number of switches in the UC is the same as the number of nodes in  $U_n^{(k)}(\Gamma_2)$ , which is summarized in Table 2. We use Waksman’s permutation network  $P_W^{(k)}$  when calculating the size of the UC, however, even with the lower bound  $P_1^{(k)}$ , for  $k \in \{5, 6, 7, 8\}$  we have the respective leading terms  $\{4.824, 4.900, 5.190, 5\}$ , which are larger than 4.75 for  $k = 4$ . The last column of Table 2 shows that the smallest UC sizes are achieved in order by Zhao et al.’s optimized UC  $\text{UC}^{\text{Zhao et al.-4}}$ , Valiant’s 4-way ( $k = 4$ )  $\text{UC}^{\text{Valiant-4}}$  and 2-way UCs ( $k = 2$ )  $\text{UC}^{\text{Valiant-2}}$ .

*Depth.* The depths of the EUG and of the UC depend only on the depth of the main skeleton, not on the subgraphs, since the longest path is between  $p_1$  and  $p_n$  in the outest skeleton. Therefore, the asymptotic depths of EUG  $U_n^{(k)}(\Gamma_1)$  and the corresponding UC are calculated as  $\frac{\text{depth}(B^{(k)})}{k}$ , as shown in the last column of Table 3. With the lower bound  $P_1^{(k)}$  for  $k \in \{5, 6, 7, 8\}$ , we have the respective leading terms  $\{4, 4, 4.14, 4\}$ , which are larger than for  $k = 2$  and  $k = 4$ . The UC depth is minimal for Valiant’s 2-way  $\text{UC}^{\text{Valiant-2}}$  ( $k = 2$ ), followed by Zhao et al.’s 4-way UC  $\text{UC}^{\text{Zhao et al.-4}}$  and Valiant’s 4-way  $\text{UC}^{\text{Valiant-4}}$  ( $k = 4$ ) as shown in Table 3.

## 6.2. Concrete Size and Depth of UCs

In this section, we consider formulae for the concrete sizes and depths of Valiant’s UCs, i.e.,  $UC^{\text{Valiant-2}}$  and  $UC^{\text{Valiant-4}}$  [66], Zhao et al.’s method  $UC^{\text{Valiant-4}}$  [72], and our hybrid universal circuits  $UC^{\text{H(Valiant-2,4)}}$  [31] and  $UC^{\text{H(Valiant-2, Zhao et al.-4)}}$ . Beforehand, we describe two optimizations.

### 6.2.1. Optimization for Fanin-1 Nodes

We observe that in  $U_n^{(k)}(\Gamma_1)$  there is a fanin-1 node in the head block (cf. [31, Fig. 2c and 4e] for  $UC^{\text{Valiant-2}}$  and  $UC^{\text{Valiant-4}}$ , respectively). A similarly designed head block for Zhao et al.’s optimized  $UC^{\text{Zhao et al.-4}}$  [72] has three such fanin-1 nodes (cf. in Fig. 19a in “Appendix B”). Moreover, fanin-1 nodes exist in the base cases for a small number of poles as well [45]. These nodes are important to achieve fanin and fanout 2 of the graph, but can be replaced with wires when translated into a circuit description as described in Sect. 3.2. Since at least one such node can be ignored in each subgraph when nodes are translated into gates, this results in at least  $k \cdot \left( \sum_{i=0}^{\log_k n-1} k^i \right) \sim kn$  less gates for the universal circuit, where  $n = u + v + g$ . We include this optimization in our calculations further on. This improvement decreases the depth of the UC only by a few gates.

### 6.2.2. Optimization for Input and Output Nodes

In the skeleton of Valiant’s UC, the poles corresponding to circuit inputs need no ingoing edges and those corresponding to circuit outputs need no outgoing edges. Therefore, since  $u$ ,  $v$  and  $g$  are publicly known, we optimize by deleting nodes that become redundant while canceling the edges going to the first  $u$  (input) and coming from the last  $v$  (output) nodes. The exact number of redundant switching nodes depends on the parity or modulo 4 of  $u$ ,  $v$ ,  $n = u + v + g$ , and the  $k$ -way UC, but is  $\mathcal{O}(u + v)$  in both  $\Gamma_1(n)$  edge-universal graphs that build up the graph of the UC. This optimization also improves the depth by  $\mathcal{O}(u + v)$ .

### 6.2.3. Concrete Sizes and Depths of 4-way and 2-way UCs

We realize that based on the parity (2-way UC) and the remainder modulo 4 (4-way UC), not only the size of the outest skeleton, but also that of the smaller subgraphs can be optimized by introducing so-called head and tail blocks (cf. Sect. 3.3 and Sect. 3.4). We considered this in our 2-way UC in [45], and we now generalize the approach for  $k$ -way UCs. We provide a recursive formula for the concrete size of the optimized  $k$ -way EUG as follows. Let  $m_k$  be

$$m_k := \begin{cases} n \bmod k & \text{if } k \nmid n, \\ k & \text{if } k \mid n. \end{cases} \quad (13)$$

Then, given the designed head, body, and tail blocks (cf. [31, Figs. 2 and 4]) with sizes and depths shown in Table 4, we can compute the size by calculating the sizes of all

**Table 4.** The sizes and depths of building blocks of the 2-way and 4-way UCs (cf. Figs. 3, 5a, b on p. 12–14, [31, Figs. 2 and 4], Figs. 19a, b in “Appendix B”), including the fanin-1 optimization from Sect. 6.2.1.

Block #poles in (next) block	Head $H^{(k)}(\cdot)$				Body $B^{(k)}(\cdot)$				Tail $T^{(k)}(\cdot)$			
	4	3	2	1	4	3	2	1	4	3	2	1
<i>Size</i>												
UC <sup>Valiant-2</sup>	–	–	3	–	–	–	5	5	–	–	4	1
UC <sup>Valiant-4</sup>	13	13	12	11	19	19	18	17	14	9	4	1
UC <sup>Zhao et al.-4</sup>	11	11	11	10	18	18	18	17	14	9	4	1
<i>Depth</i>												
UC <sup>Valiant-2</sup>	–	–	3	–	–	–	6	6	–	–	4	1
UC <sup>Valiant-4</sup>	10	10	9	9	15	15	14	14	11	9	4	1
UC <sup>Zhao et al.-4</sup>	9	9	9	9	14	14	14	14	11	9	4	1

the components of the outest skeleton, and the sizes of the smaller subgraphs with the recursive formula in Eq. 14.<sup>4</sup>

$$\begin{aligned}
\text{size}(U_n^{(k)}(\Gamma_1)) &= \text{size}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{size}(B^{(k)}(k)) + \text{size}(B^{(k)}(m_k)) \\
&\quad + \text{size}(T^{(k)}(m_k)) + m_k \cdot \text{size}\left(U_{\lceil \frac{n}{k} \rceil - 1}^{(k)}(\Gamma_1)\right) \\
&\quad + (k - m_k) \cdot \text{size}\left(U_{\lfloor \frac{n}{k} \rfloor - 1}^{(k)}(\Gamma_1)\right).
\end{aligned} \tag{14}$$

As described in Sect. 3.1, a UC is constructed by means of an EUG  $U_n^{(k)}(\Gamma_2)$ , which is in turn constructed from two EUGs with fanin and fanout one,  $U_n^{(k)}(\Gamma_1)$ , by merging their poles together and thus taking them only once into consideration. When constructing a UC for circuit  $C_{u,v}^g$ , the number of inputs  $u$ , the number of outputs  $v$ , and the number of gates  $g$  with fanin and fanout 2 are public. Thus, using Valiant’s construction,  $U_n^{(k)}(\Gamma_2)$  with  $n = u + v + g$  poles is constructed, and thus, our formula for the concrete size of  $U_n^{(k)}(\Gamma_2)$  corresponding to  $C_{u,v}^g$  is

$$\text{size}(U_n^{(k)}(\Gamma_2)) = 2 \cdot \text{size}(U_n^{(k)}(\Gamma_1)) - n, \tag{15}$$

and the size of the UC is

$$\text{size}(UC_n) \leq (\text{size}(U_n^{(k)}(\Gamma_2)) - n) \cdot \text{size}(X) + g \cdot \text{size}(U), \tag{16}$$

where  $X$ ,  $Y$ , and  $U$  denote X-, Y-switching blocks and universal gates (cf. Sect. 3.2), respectively, and  $\text{size}(Y) \leq \text{size}(X) \leq \text{size}(U)$ .

<sup>4</sup>We note that for  $k \geq 3$ , there exist  $H^{(k)}(k-1), \dots, H^{(k)}(1)$  blocks. These are used for only one  $n$ , e.g.,  $H^{(k)}(1)$  when  $n = k+1$ , and  $H^{(k)}(k-1)$  when  $n = 2k$ . For simplicity, we consider these as special recursion base numbers in our calculations, but the formula can be adapted to include these as well.

The depth of a  $k$ -way UC also depends on  $m_k$ , the head, tail and body blocks (cf. [31, Figs. 2 and 4]), but not on the subgraphs. Thus, it is calculated using the formula in Eq. 17.

$$\begin{aligned} \text{depth}(U_n^{(k)}(\Gamma_1)) &= \text{depth}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{depth}(B^{(k)}(k)) \\ &\quad + \text{depth}(B^{(k)}(m_k)) + \text{depth}(T^{(k)}(m_k)). \end{aligned} \quad (17)$$

Since  $\text{depth}(U_n^{(k)}(\Gamma_2)) = \text{depth}(U_n^{(k)}(\Gamma_1))$ , the depth of the UC is

$$\text{depth}(UC_n) \leq (\text{depth}(U_n^{(k)}(\Gamma_2)) - n) \cdot \text{depth}(X) + g \cdot \text{depth}(U), \quad (18)$$

where  $\text{depth}(Y) \leq \text{depth}(X) \leq \text{depth}(U)$ .

#### 6.2.4. Concrete Size and Depth of Our 2/4 Hybrid UC

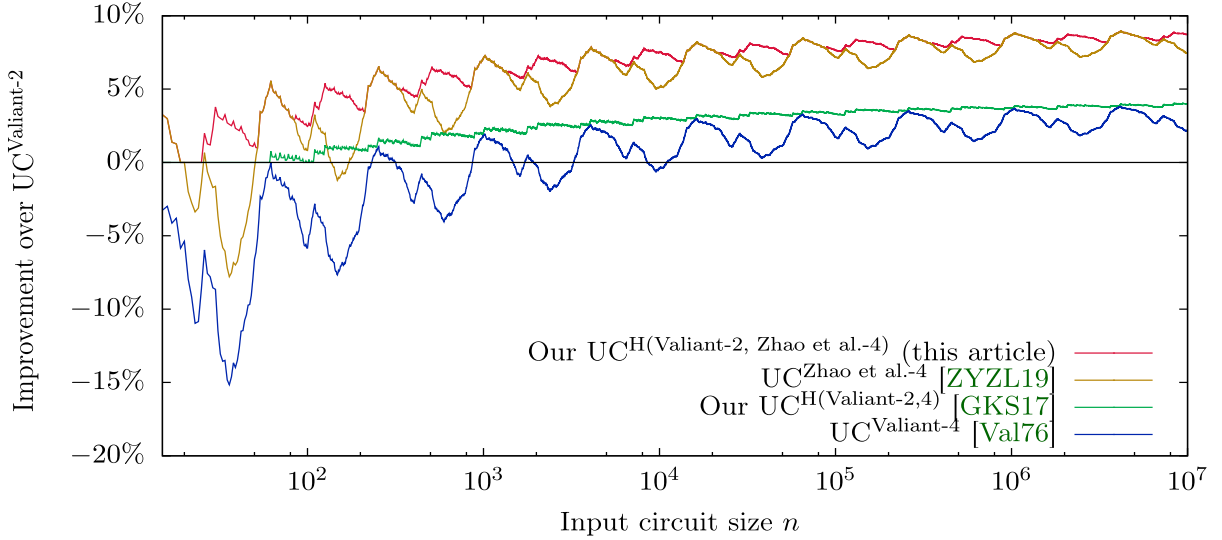
In Sect. 5.3, we provide a construction for minimizing the concrete size of the resulting 2/4 hybrid UC. The construction chooses at each step the skeleton that results in the smallest size. We provide the formula for determining its size using a dynamic programming algorithm in Eq. 19.  $\text{size}(H^{(k)}(i))$ ,  $\text{size}(T^{(k)}(i))$  and  $\text{size}(B^{(k)}(i))$  are values from Table 4 for  $k = 2$  and  $k = 4$ . Its depth is the depth of the outermost skeleton, either of the 4-way or 2-way UC, depending on which is chosen first.

$$\begin{aligned} \text{size}(U_n^{\text{hybrid}(K)}(\Gamma_1)) &= \min \left( \text{size}(H^{(k)}(k)) + \left(\left\lceil \frac{n}{k} \right\rceil - 3\right) \cdot \text{size}(B^{(k)}(k)) \right. \\ &\quad + \text{size}(B^{(k)}(m_k)) + \text{size}(T^{(k)}(m_k)) \\ &\quad + m_k \cdot \text{size}\left(U_{\left\lceil \frac{n}{k} - 1 \right\rceil}^{\text{hybrid}(K)}(\Gamma_1)\right) \\ &\quad \left. + (k - m_k) \cdot \text{size}\left(U_{\left\lfloor \frac{n}{k} - 1 \right\rfloor}^{\text{hybrid}(K)}(\Gamma_1)\right) \right); \\ &\quad k \in K = \{2, 4\}. \end{aligned} \quad (19)$$

#### 6.2.5. Improvements in Size over Valiant's 2-way UC

Figure 9 shows the concrete improvement in percentage of  $UC^{\text{Valiant-4}}$  and  $UC^{\text{Zhao et al.-4}}$  over  $UC^{\text{Valiant-2}}$  up to ten million nodes in the simulated input circuit. All reported averages are for the interval  $n \in \{15, \dots, 10^7\}$ . From the asymptotic leading factors in Table 2, we expect an improvement of up to 5% for  $UC^{\text{Valiant-4}}$  and up to 10% for  $UC^{\text{Zhao et al.-4}}$ . In Table 5, we depict the minimum, average, and maximum improvement compared to the asymptotic improvement in the interval  $n \in \{2, \dots, 10^7\}$ . For the smallest  $n$  values ( $n \leq 15$ ),  $UC^{\text{Valiant-2}}$  is better than both 4-way UCs. However, with growing values of  $n$ , the 4-way UCs are better, except for some short intervals as shown in Fig. 9. However, Valiant's and Zhao et al.'s 4-way UCs always outperform Valiant's 2-way UC for  $n \geq 10\,996$  and  $n \geq 172$ , respectively, the average improvement being 2.97% and 7.65%, and the biggest improvement being 3.78% and 8.88%.

The improvement of our  $UC^{\text{H(Valiant-2,4)}}$  and  $UC^{\text{H(Valiant-2, Zhao et al.-4)}}$  (cf. Sect. 5.3) is depicted in the same Fig. 9 and summarized in Table 5. For some  $n$  values, our



**Fig. 9.** Improvement in size in percentage of our 2/4 hybrid, the 4-way UCs of [66, 72] over Valiant’s 2-way UC for  $15 \leq n \leq 10^7$  with logarithmic  $x$  axis. We note that the different graphs are in the same order as in the legend.

**Table 5.** Minimum, average, maximum, and expected asymptotic improvement in size of our 2/4 hybrid and the 4-way UCs of [66, 72] over Valiant’s 2-way UC in the range  $15 \leq n \leq 10^7$ .

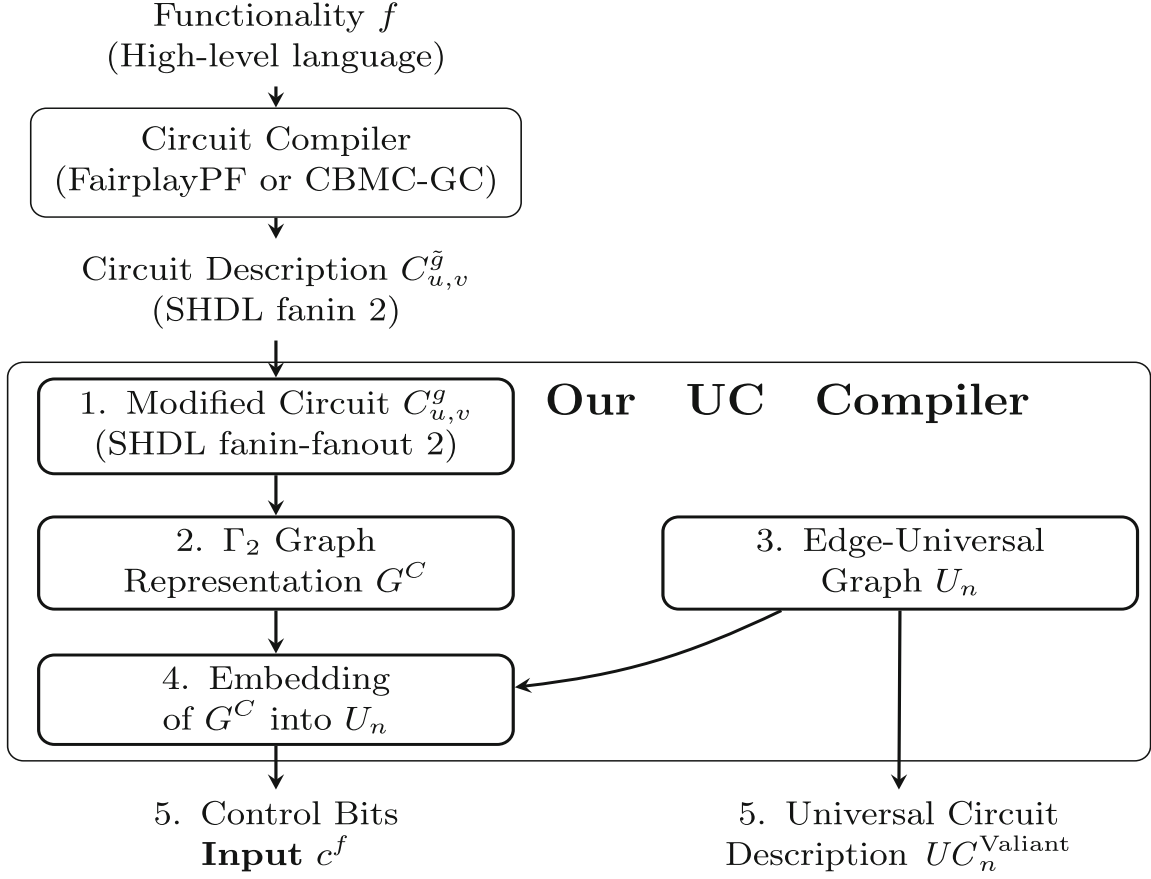
UC	Reference	Minimum (%)	Average (%)	Maximum (%)	Asymptotic (%)
$UC^{\text{Valiant-4}}$	[66]	− 34.78	2.97	3.78	5
$UC^{\text{H(Valiant-2,4)}}$	[31]	0	3.41	4.00	5
$UC^{\text{Zhao et al.-4}}$	[72]	− 26.09	7.65	8.88	10
$UC^{\text{H(Valiant-2, Zhao et al.-4)}}$	(This article)	0	7.71	8.88	10

hybrid UCs achieve the same size as the 2-way or corresponding 4-way UCs, but due to their nature, their improvement is always nonnegative, and greater than or equal to the improvement achieved by the 4-way UC. Moreover, in most cases our hybrid UCs result in better sizes than the underlying 4-way UC, which means that some subgraphs are created for an  $n$  for which the 2-way UC is smaller. The overall improvement over  $UC^{\text{Valiant-2}}$  for all  $n \in \{2, \dots, 10^7\}$  values of our  $UC^{\text{H(Valiant-2,4)}}$  is on average 3.41% and at most 4.00%, and for our  $UC^{\text{H(Valiant-2, Zhao et al.-4)}}$  is on average 7.71% and at most 8.88%.

We note that our hybrid UC can also be used to reduce the depth of the UC by utilizing the 2-way UC,  $UC^{\text{Valiant-2}}$ , in the first step of the construction. This results in the smallest asymptotic depth  $\sim 3n$  (cf. Table 3).

## 7. Implementation and Evaluation of Our UC Compiler

In this section, we detail the challenges faced while demonstrating the practicality of Valiant’s and Zhao et al.’s universal circuits. We show how to construct a UC and program it according to a standard circuit description. We validate our results with a



**Fig. 10.** Our universal circuit compiler.

practical implementation that, upon receiving a fanin-2 circuit  $C_{u,v}^g$  as input, outputs the corresponding 2-way or 4-way UC  $UC^{\text{Valiant-2}}$ ,  $UC^{\text{Valiant-4}}$  or  $UC^{\text{Zhao et al.-4}}$  and its programming  $c^f$ . We have provided the first implementation of Valiant’s 2-way UC of size  $\sim 5n \log_2 n$  in [45] and implemented Valiant’s 4-way UC of smaller size  $\sim 4.75n \log_2 n$  in a modular way in [31].

In this work, we extend our implementation with the modular 2-way UC and include the optimized 4-way UC of Zhao et al. [72] with size  $\sim 4.5n \log_2 n$ . We then combine the modular 2-way UC with both 4-way UCs in an implementation of our hybrid UC proposed in [31] and Sect. 5.3, i.e.,  $UC^{\text{H(Valiant-2,4)}}$  and  $UC^{\text{H(Valiant-2, Zhao et al.-4)}}$ , respectively. Moreover, we provide a prototype implementation of our scalable 4-way UC from Sect. 5.4, which can be generalized to both the 2-way UC and Zhao et al.’s improvement.

### 7.1. UC Compiler

The architecture of our UC compiler is depicted in Fig. 10. In this section, we briefly describe its different artifacts and its use of the Fairplay [51] or CBMC-GC [10,23] frameworks as a frontend. For a more detailed description, the reader is referred to [45]. Our implementation is available online at <https://crypto.de/code/UC>.



1. *Compiling Input Circuits from High-Level Functionality.* We can use the Fairplay compiler [11,51] with the FairplayPF extension [44] or the CBMC-GC compiler [10,23] to translate the functionality described in a high-level language to the Fairplay circuit description called Secure Hardware Definition Language (SHDL). These compilers output a circuit  $C_{u,v}^{\tilde{g}}$  with fanin 2, which is required for all UCs. However, due to Valiant's design, the input circuit  $C_{u,v}^{\tilde{g}}$  to our UC compiler has to have fanout 2 as well, i.e., the outputs of all gates and inputs can only be used as the input of at most two subsequent gates. This can be achieved using copy gates such that instead of  $\tilde{g}$  gates, we have  $\tilde{g} \leq g \leq 2\tilde{g} + v$  fanout-2 gates (cf. Sect. 2.2). We give concrete examples in [45] on how this conversion affects the size of practical circuits and show that in most cases, the resulting number of gates remains significantly below the upper bound  $2\tilde{g} + v$ .
2. *Obtaining the  $\Gamma_2(n)$  Graph  $G$  of the Circuit  $C_{u,v}^g$ .* As next step, we transform circuit  $C_{u,v}^g$  into a  $\Gamma_2(n)$  graph  $G = (V, E)$  with  $n = u + v + g$  (cf. Sect. 3.1). This can directly be generated as described in Sect. 2.2: With the number of inputs  $u$ , outputs  $v$ , and gates  $g$  in circuit  $C_{u,v}^g$ ,  $G$  has  $n$  nodes and the wires are represented as edges in the graph. Then, we define a topological order  $\eta^G$  on the nodes of  $G$  such that every input node  $v_i$  has a topological order of  $1 \leq \eta^G(v_i) \leq u$  and every output node  $v_j$  is labeled with  $u + g + 1 \leq \eta^G(v_j) \leq u + v + g$ . Since  $C_{u,v}^g$  has fanin and fanout 2, the resulting graph  $G$  is in  $\Gamma_2(n)$ , where  $n = u + v + g$ . It is possible in the modified SHDL circuit description that an internal value becomes two times the first or two times the second input of gates. Therefore, when a value is the second time the same input to a gate (i.e., first or second), both the two inputs and the two middle bits of the function table of the gate must be reversed (i.e., to compute  $f(\text{in}_1, \text{in}_2)$  instead of  $f(\text{in}_2, \text{in}_1)$ ) for the correct programming of the UC in Step 5.
3. *Generating Edge-Universal Graph  $U_n^{(\ell)}(\Gamma_2)$  or  $U_n^{\text{hybrid}(K)}(\Gamma_2)$  for  $\Gamma_2(n)$  graphs, where  $\ell \in \{2, 4\}$  and  $K = \{2, 4\}$ .* An EUG  $U_n^{(\ell)}(\Gamma_2)$  or  $U_n^{\text{hybrid}(K)}(\Gamma_2)$  is constructed by creating two instances of  $U_n^{(\ell)}(\Gamma_1)$  or  $U_n^{\text{hybrid}(K)}(\Gamma_1)$ , respectively, as described in Sect. 3.1. The two instances get merged to  $U_n^{(\ell)}(\Gamma_2)$  so that one builds the left inputs and outputs and the other builds the right inputs and outputs of the gates (based on the two-coloring of  $G$ ). For efficiency reasons, we directly generate the merged edge-universal graph, i.e., an EUG for  $\Gamma_2(n)$ , with the poles as common nodes. We partly include our optimization for the input and output nodes from Sect. 6.2.2<sup>5</sup> and Valiant's optimizations for the base cases  $n \in \{2, 3, 4\}$ , but do not consider Valiant's optimizations for  $n \in \{5, 6\}$  [66]. Knowing the number of input bits  $u$ , the number of gates  $g$ , and the number of output bits  $v$ , we construct the corresponding edge-universal graph  $U_n^{(\ell)}(\Gamma_2)$ , where  $n = u + v + g$ . We note that no knowledge is necessary about the topology or the gate tables in circuit  $C$  for this step.
4. *Programming  $U_n^{(\ell)}(\Gamma_2)$  and  $U_n^{\text{hybrid}(K)}(\Gamma_2)$  According to an Arbitrary  $\Gamma_2(n)$  Graph.* We edge-embed graph  $G$  into  $U_n^{(\ell)}(\Gamma_2)$  as described in Sect. 4 and into our

<sup>5</sup>We delete edges coming into inputs and going out from outputs. Due to this, some nodes are removed due to our fanin-1 optimization from Sect. 6.2.1 when translated into a UC.



hybrid  $U_n^{\text{hybrid}(K)}(\Gamma_2)$  with  $K = \{2, 4\}$  as described in Sect. 5.3.  $G$  is partitioned into two  $\Gamma_1(n)$  graphs  $G_1$  and  $G_2$  which are embedded into the two EUGs  $U_n^\ell(\Gamma_1)_1$  and  $U_n^\ell(\Gamma_1)_2$ . Valiant proved in [66] that any topologically ordered  $\Gamma_1(n)$  graph can be edge-embedded in an EUG  $U_n^\ell(\Gamma_1)$  (cf. Sect. 3.1). We perform the embedding as described in Sect. 4 for Valiant’s 2-way and 4-way EUGs in Listing 1. The difference when using Zhao et al.’s improvement [72] is the block edge-embedding described in Sect. 4.1. Here, we utilize a lookup table derived from the computer generated proof of Zhao et al. [72] that maps the *in* and *out* vectors as defined in Sect. 4.1 into the programming bits of the block, i.e., can be used as block edge-embedding along with the recursion point edge-embedding described in Sect. 4.2. We edge-embed  $G_1$  and  $G_2$  into our 2/4-hybrid EUGs  $U_n^{\text{hybrid}(K)}(\Gamma_1)_1$  and  $U_n^{\text{hybrid}(K)}(\Gamma_1)_2$  as described in Sect. 5.3. When the edge-embedding is finished, we define the control bits of the programmable blocks (universal gates and switches) as described in Sect. 3.2.

5. *Generating the Output Circuit Description and the Programming of the Universal Circuit.* After embedding the graph of the simulated circuit into the edge-universal graph  $U_n(\Gamma_2)$ , we write the resulting circuit in a file using our generic UC description. In the edge-universal graph, each node stores the control bit resulting from the edge-embedding (control bit  $c$  of the corresponding universal switch in Sect. 3.2) and each pole corresponding to a gate stores four bits (the four control bits of the function table of the corresponding gate in the original circuit  $C_{u,v}^g, c_0, c_1, c_2, c_3$  in Eq. 1, their order possibly changed in Step 2). Thus, after topologically ordering  $U_n(\Gamma_2)$ , one can directly write out the gate identifiers into a circuit file  $UC$  and the control bits to a programming file  $c^f$ . We include our optimization from Sect. 6.2.1 and ignore extra nodes with fanin 1 when the graph is translated into a UC description. This improves the size of the recursion bases for  $n = \{4, 5, 6\}$  as well as of the head blocks [31, Fig. 2c and Fig. 4e] and Fig. 19a in “Appendix B.”

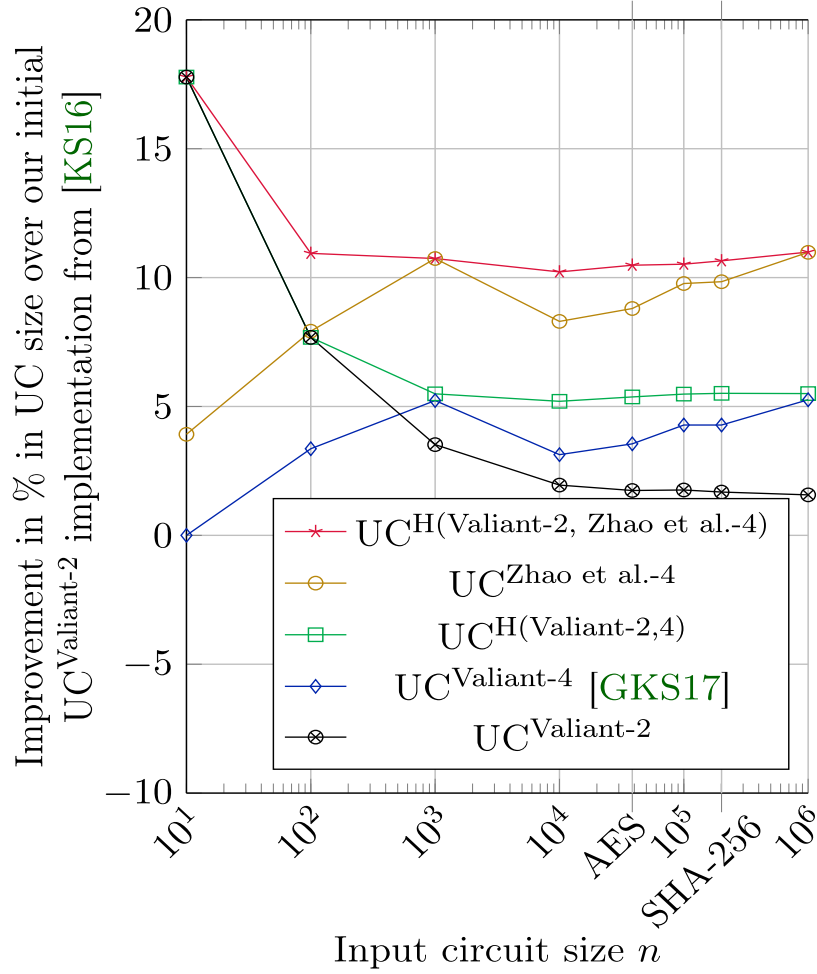
Our circuit description format is generic, i.e., consists of universal switches and universal gates. Therefore, any framework can be adapted to use them, independently from if it is interpreted as a Boolean or arithmetic UC. We start with enumerating the client input wires as  $C \ 0 \ 1 \ \dots \ u - 1$ . As a reminder, the  $\mathcal{O}(n \log n)$  server input wires are in the programming file  $c^f$ . In the UC, we have universal gates denoted by  $U$ , universal switches denoted by  $X$  or  $Y$  depending on the number of outputs ( $X$  with two outputs and  $Y$  with one):

$$U \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \tag{20}$$

$$X \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \quad \text{out}_2 \tag{21}$$

$$Y \quad \text{in}_1 \quad \text{in}_2 \quad \text{out}_1 \tag{22}$$

denotes that wire  $\text{out}_1$  (and possibly  $\text{out}_2$ ) is coming from a gate with input wires  $\text{in}_1$  and  $\text{in}_2$ . The control bits are not represented in the circuit format, but for each universal gate we save a four-bit number representing the control bits and for each universal switch we store the control bit in the programming file  $c^f$ . The output nodes are outputs of  $Y$  universal switches and are marked in the end of the file as  $O \ 0_1 \ 0_2 \ \dots \ 0_v$ . The



**Fig. 11.** Improvement in percentage of the UC sizes (number of switches) of our UC implementation of Valiant’s 4-way  $UC^{\text{Valiant-4}}$  from [31] and our novel implementations including a modular version of Valiant’s 2-way  $UC^{\text{Valiant-2}}$ , Zhao et al.’s improved block  $UC^{\text{Zhao et al.-4}}$  and hybrid constructions  $UC^H(\text{Valiant-2,4})$  and  $UC^H(\text{Valiant-2, Zhao et al.-4})$  over our implementation of Valiant’s 2-way UC from [45].

circuit and its programming are given in plain text files as shown in Listings 3 and 4 in “Appendix C.”

## 7.2. Experimental Evaluation

We ran all experiments for our UC compiler on a Desktop PC, equipped with an Intel Core i7-4790 CPU with 3.6 GHz and 32 GB RAM, and provide our results in this section. We performed experiments for circuit sizes  $n \in \{10, 100, \dots, 1\,000\,000\}$  as well as with notable circuits from [65] such as the AES-128 circuit without key expansion with size  $n = 38\,518$  and the SHA-256 circuit with size  $n = 201\,206$ . We note that these sizes are for the circuits transformed to have fanin and fanout 2 as described in Sect. 2.2 and in [45, Table 1].

*Circuit Sizes* (Fig. 11). We first compare the circuit sizes of our implementations that slightly differ from the expected sizes shown in Sect. 6. Our initial 2-way  $UC^{\text{Valiant-2}}$  implementation from [45] included the recursion bases for 1, 2, and 3 nodes and, however, did not include those proposed by Valiant [66] optimized for 4, 5, and 6 nodes. It included

both size optimizations described in Sects. 6.2.1 and 6.2.2. In Fig. 11, we show the improvement over our  $\text{UC}^{\text{Valiant-2}}$  implementation from [45] in percentage of the number of switches of our later, more modular UC implementations presented in this article and in [31]. We note that the number of universal gates is the same for all implementations, i.e., the number of gates in the original circuits  $g$ .

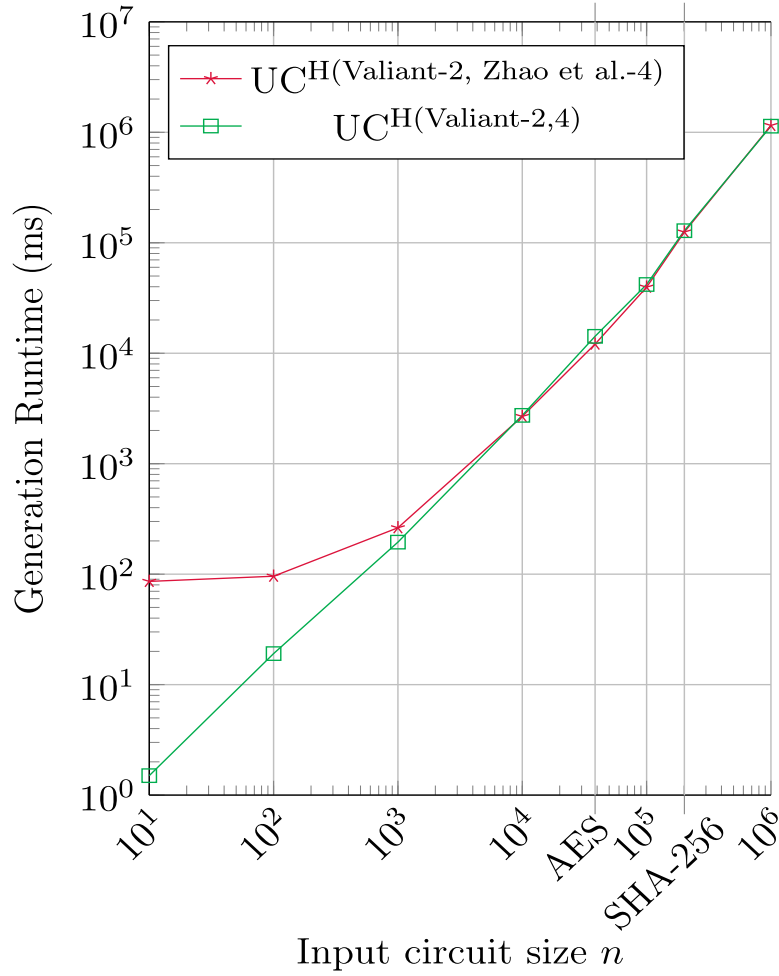
Our modular 4-way  $\text{UC}^{\text{Valiant-4}}$  implementation from [31] additionally included the recursion base with 4 nodes and, however, only partly included the optimization described in Sect. 6.2.2 concerning the input and output nodes. The edges directed into the inputs and out of the outputs are also removed which results in smaller sizes due to the thus redundant nodes, however, not all unnecessary connections are deleted. This, however, incurs only a small overhead of at most  $\mathcal{O}(u + v)$ . As we can observe in Fig. 11 and as expected (cf. Table 5 on p. 32), this implementation improved by around 5% over our implementation from [45].

In this article, we have first implemented the modular version of Valiant’s 2-way  $\text{UC}^{\text{Valiant-2}}$  where inherently we use the optimized recursion base with 4 nodes as well. An around 1.5-2% improvement can be observed over our non-modular implementation from [45]. Using this and our modular 4-way  $\text{UC}^{\text{Valiant-4}}$ , we have implemented our hybrid  $\text{UC}^{\text{H(Valiant-2,4)}}$  using Valiant’s 2-way and 4-way UCs as proposed in [31]. This implementation has a more steady improvement of at least 5% for most tested circuit sizes. Moreover, we also implemented the optimized  $\text{UC}^{\text{Zhao et al.-4}}$  proposed in [72], who have proved that their optimized block is universal by giving the programming for all possible path combinations in the block. We use this proof to generate a lookup table file for our implementation, which contains a mapping from any possible input–output vector (cf. Sect. 4.1) and the corresponding programming bits for the block. The generation of this lookup table is a one-time precomputation cost and takes around 82 seconds. In subsequent runs of the UC compiler, this overhead is no longer needed and a file of size 1.08 MB is read which takes only about 80 milliseconds. Thereafter, the expected gain of around 10% can be observed over our 2-way  $\text{UC}^{\text{Valiant-2}}$  implementation from [45]. Moreover, the hybrid variant with this construction, i.e.,  $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ , achieves an at least 10% improvement for all our example circuits.

In Table 6, we show the concrete number of switches of the smallest UCs generated with  $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$  as well as the sizes of the resulting UC and programming files. The universal circuit for  $n = 1$  million gates has around 76 million switches and additionally around 1 million universal gates (which, in the PFE setting, results in a total of about 77 million AND gates for Yao’s garbled circuit protocol and 79 million AND gates for the GMW protocol). The corresponding file for the UC has size 2.8 GB, and the programming file has size 0.15 GB.

*Runtime* (Fig. 12). To compare the runtime of our UC implementation with that of the UC compiler of [45], we ran the same experiments on the same platform using our novel implementations for  $\text{UC}^{\text{Valiant-2}}$ ,  $\text{UC}^{\text{Zhao et al.-4}}$ ,  $\text{UC}^{\text{H(Valiant-2,4)}}$ , and  $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$ . Runtimes are reported as averages from 10 executions. The differences in runtimes for the different constructions are not significant, and therefore, we only depict the runtimes of our hybrid implementations  $\text{UC}^{\text{H(Valiant-2,4)}}$  and  $\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$  in Fig. 12.

The runtimes of our modular  $\text{UC}^{\text{Valiant-2}}$  and  $\text{UC}^{\text{Valiant-4}}$  implementations are very similar to those of  $\text{UC}^{\text{H(Valiant-2,4)}}$ , the latter of which becomes best for larger circuits



**Fig. 12.** Comparison of the runtime of our hybrid UC implementations using either Valiant’s 2-way and 4-way UCs or Valiant’s 2-way UC with Zhao et al.’s improved block. We note that the runtime of  $UC^{\text{Zhao et al.-4}}$  only slightly differs from that of  $UC^H(\text{Valiant-2, Zhao et al.-4})$ , and the runtimes of  $UC^{\text{Valiant-2}}$  and  $UC^{\text{Valiant-4}}$  only slightly differ from that of  $UC^H(\text{Valiant-2,4})$ , and therefore, we omit them from the figure.

**Table 6.** Size of our smallest UCs generated with  $UC^H(\text{Valiant-2, Zhao et al.-4})$ , i.e., its number of switches, the sizes of the UC, and programming files.

Input circuit size $n$	10	100	1000	10,000	38,518 (AES)	100,000	201,206 (SHA-256)	1,000,000
Size (#switches)	45	1719	31,667	462,667	2,119,836	6,147,387	13,277,772	76,484,267
UC file (KB)	0.6	36	794	13,473	68,730	207,789	473,915	2,936,852
Prog. file $c^f$ (KB)	0.1	4	65	933	4224	12,300	26,391	152,314

(i.e., our examples with  $n \geq 10,000$ ). The runtimes of  $UC^{\text{Zhao et al.-4}}$  are only slightly lower than those of our hybrid  $UC^H(\text{Valiant-2, Zhao et al.-4})$ , both of which include a one-time overhead of around 80 milliseconds for reading in our lookup table of size 1.08 MB for each possible block programming [72]. However, this one-time expense is only significant for small circuits as can be observed in Fig. 12, and  $UC^H(\text{Valiant-2, Zhao et al.-4})$  becomes faster than  $UC^H(\text{Valiant-2,4})$  for our examples with  $n \geq 10,000$ . The runtime of our original 2-way  $UC^{\text{Valiant-2}}$  from [45] was slightly better due to its handling of the

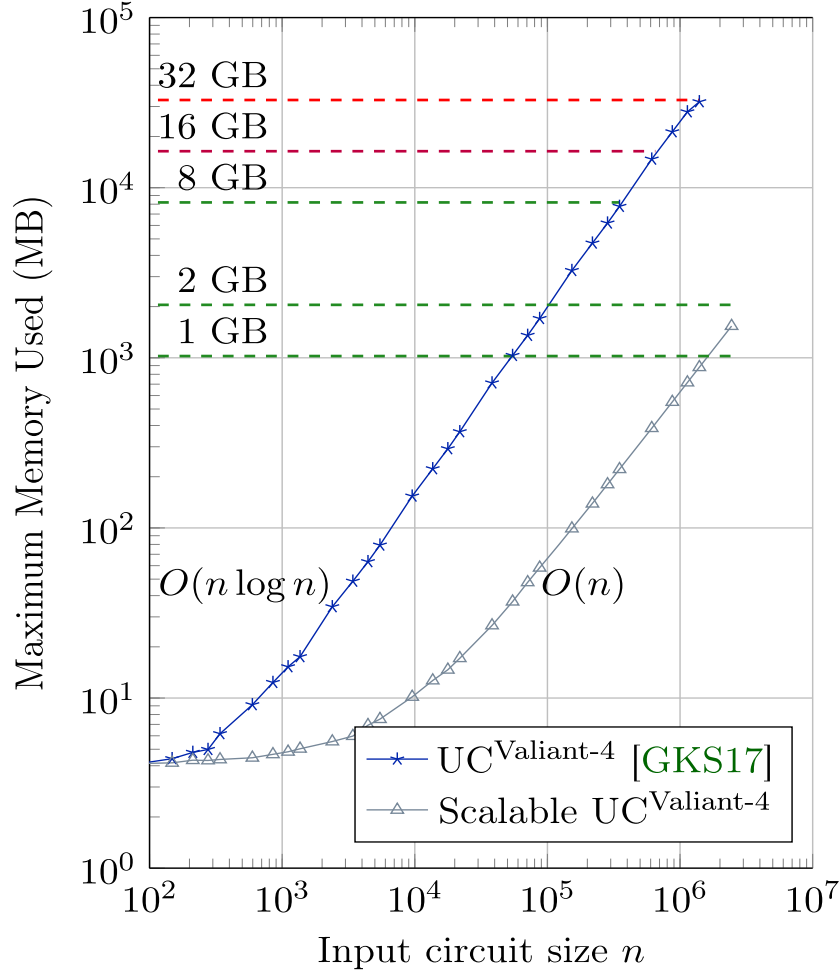
UC as one big block. However, it also becomes worse than  $UC^{H(\text{Valiant-2, Zhao et al.-4})}$  for our largest examples SHA-256 and the circuit for one million gates due to the gain in the size that results in a less complex embedding. For instance, it takes about 12 s to generate the smallest  $UC^{H(\text{Valiant-2, Zhao et al.-4})}$  with our new implementation for AES-128, while our original implementation for  $UC^{\text{Valiant-2}}$  took 9.4 s. Our largest examples SHA-256 and a circuit with one million gates were generated and programmed in 2.1 and 18.6 min, respectively. The runtimes are high for these large examples; however, they are generally a one-time precomputation expense in most application scenarios such as private function evaluation (cf. Sect. 1.1).

*Scalable 4-way UC Implementation* (Figs. 13, 14). We also implemented our scalable 4-way UC generation algorithm presented in Sect. 5.4. We note that our implementation only includes  $H^i$ ,  $T_x^i$  and  $B_x^i$  for  $i = 0, 1, 2, 3$  and  $x = 4$  and does not include the optimized versions for  $x = 1, 2, 3$  which we leave as future work. Moreover, we include the base cases for  $n = 1, 2, 3$  but not that for  $n = 4$ . This is due to the fact that a lot of engineering effort would be required for including the other options as well and our work is only a proof-of-concept implementation of our method presented in Sect. 5.4. Therefore, we test circuits with specific sizes where none of the other blocks or base case are required, i.e., where all subgraphs at each recursion step have 4 nodes in the tail block and the base case with  $n = 4$  is not needed. Currently, for generating UCs for different sizes, one would need to pad the original circuit with dummy gates to an allowed size. Our aim was to improve the memory consumption of the UC generation (and programming) algorithm, while keeping the price paid in runtime as low as possible. The number of files created is the number of subgraphs in the UC, which is necessary for efficient scalable programming of the UC.

We show that our scalable UC generation implementation provides the expected improvement in memory usage by comparing our scalable  $UC^{\text{Valiant-4}}$  implementation to our implementation from [31]. We depict in Fig. 13 the memory usage of the generation algorithm with growing input circuit sizes on a machine with 32 GB RAM memory. As can be seen in the figure, instead of holding the whole UC of size  $\mathcal{O}(n \log n)$  in memory, we indeed hold only  $\mathcal{O}(n)$  information in memory at each step. When using 1 GB, 8 GB, and 32 GB of memory, we can generate a UC for over  $27\times$ ,  $28\times$ , and  $29\times$  larger input circuit sizes  $n$ , respectively. Moreover, as can be observed in Fig. 14, the runtime of the resulting scalable UC generation is only around  $4\times$  that of the  $UC^{\text{Valiant-4}}$  implementation of [31]. This difference is becoming smaller with increasing  $n$  due to the fact that the implementation of [31] is running short on memory and starts swapping to disk. Our experiments show that while reducing the memory requirements of our UC generation for  $UC^{\text{Valiant-4}}$ , we keep the runtime asymptotically the same (cf. Fig. 14). Moreover, the required storage capacity is also  $\mathcal{O}(n \log n)$  as before, since the additionally stored data at each step are at most  $\mathcal{O}(n)$ , cf. Sect. 5.4.

## 8. Toolchain for Private Function Evaluation

Secure function evaluation (SFE) allows two parties to jointly compute a public function on their private inputs, without revealing anything to each other apart from the output of the computation. As it is probably the most prominent application of UCs



**Fig. 13.** Comparison of the maximum memory used between our per-block and [31]’s UC generation. [31]’s implementation runs out of 32 GB of memory for  $n > 1\,398\,100$  nodes.

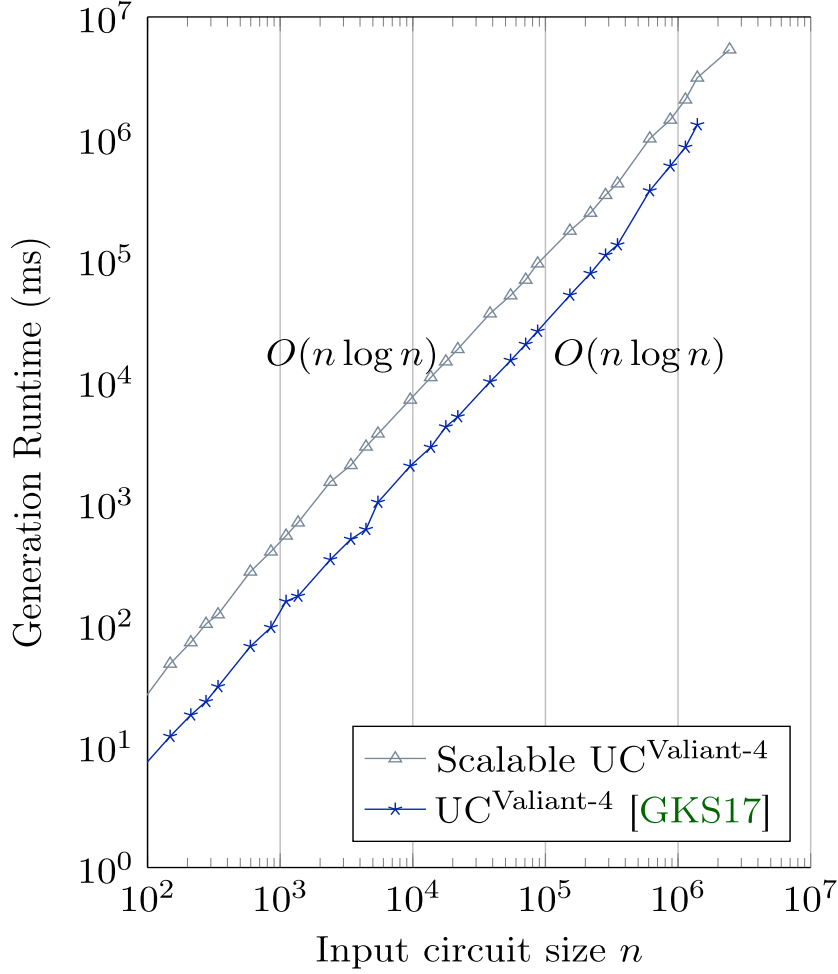
(cf. Sect. 1.1), we implement private function evaluation (PFE) using SFE of a Boolean universal circuit. In this scenario, one of the parties holds its input  $x$  and the other party holds the programming  $c^f$  corresponding to a private function  $f$  that allows the UC to compute  $UC(x, c^f) = f(x)$ . We note that the UC (with control bits for the universal gates and switches) can be publicly generated.

We have created a novel toolchain for private function evaluation (PFE) in [45], using the ABY framework for SFE (secure against semi-honest adversaries) as backend of our UC compiler. ABY implements state-of-the-art optimizations of Yao’s garbled circuit protocol [69, 70] and the GMW protocol [32]. We emphasize that our tool for constructing and programming UC is generic and can easily be adapted to other secure computation frameworks or other applications of UCs listed in Sect. 1.1.

### 8.1. Extension of the ABY Framework

We adapt the ABY secure two-party computation framework [19] for securely evaluating universal circuits. We realize the universal circuit building blocks (universal gates and switches) with a number of AND and XOR gates, which is the functionally complete





**Fig. 14.** Comparison of the runtime of our per-block and [31]’s UC generations for up to about  $n = 2,446,000$  nodes, which fails with [31]’s UC generation and 32 GB of memory.

set of logical gates that ABY uses. Since XOR gates can be evaluated for free in the underlying protocols for secure function evaluation due to the free-XOR optimization [43], from here on, we study the *AND-size* ( $\text{size}^{\text{AND}}$ ) and *AND-depth* ( $\text{depth}^{\text{AND}}$ ) of UCs, i.e., the number of AND gates and the maximum number of AND gates on the longest path, respectively. For other applications, however, the *total sizes and depths* of the UCs with respect to both AND and XOR gates are relevant. We implement universal gates and switches optimized for PFE and therefore use few AND gates, and only (free) XOR gates alongside it.  $X$  and  $Y$  gates are obtained as shown in [43]

$$\text{out}_1 = Y(\text{in}_1, \text{in}_2; c) = (\text{in}_1 \oplus \text{in}_2)c \oplus \text{in}_1 \quad (23)$$

$$(\text{out}_1, \text{out}_2) = X(\text{in}_1, \text{in}_2; c) = (e \oplus \text{in}_1, e \oplus \text{in}_2) \text{ with } e = (\text{in}_1 \oplus \text{in}_2)c \quad (24)$$

with  $\text{size}^{\text{AND}}(Y) = \text{size}^{\text{AND}}(X) = \text{depth}^{\text{AND}}(Y) = \text{depth}^{\text{AND}}(X) = 1$  for both universal switches. In case the SFE implementation uses Yao’s garbled circuit protocol [70], both  $\text{size}^{\text{AND}}(U) = 1$  and  $\text{depth}^{\text{AND}}(U) = 1$ , due to the fact that in some garbling schemes (such as in the case of garbled 3-row reduction (GRR3) [55]) the evaluator does not learn the type of the evaluated gate. Therefore, a universal gate can be imple-



mented using only one 2-input non-XOR gate [60]. For other SFE protocols such as GMW where this optimization is not possible, our efficient implementation of generic universal gates uses  $Y$  gates yielding

$$\text{out}_1 = U(\text{in}_1, \text{in}_2; c_0, c_1, c_2, c_3) = Y[Y(c_0, c_1; \text{in}_2), Y(c_2, c_3; \text{in}_2); \text{in}_1] \quad (25)$$

with  $\text{size}^{\text{AND}}(U) = 3$  and  $\text{depth}^{\text{AND}}(U) = 2$ . We note that the implementation of switches and universal gates might look very different when other 2-input Boolean gates can also be used, e.g., when other size metrics are to be minimized.

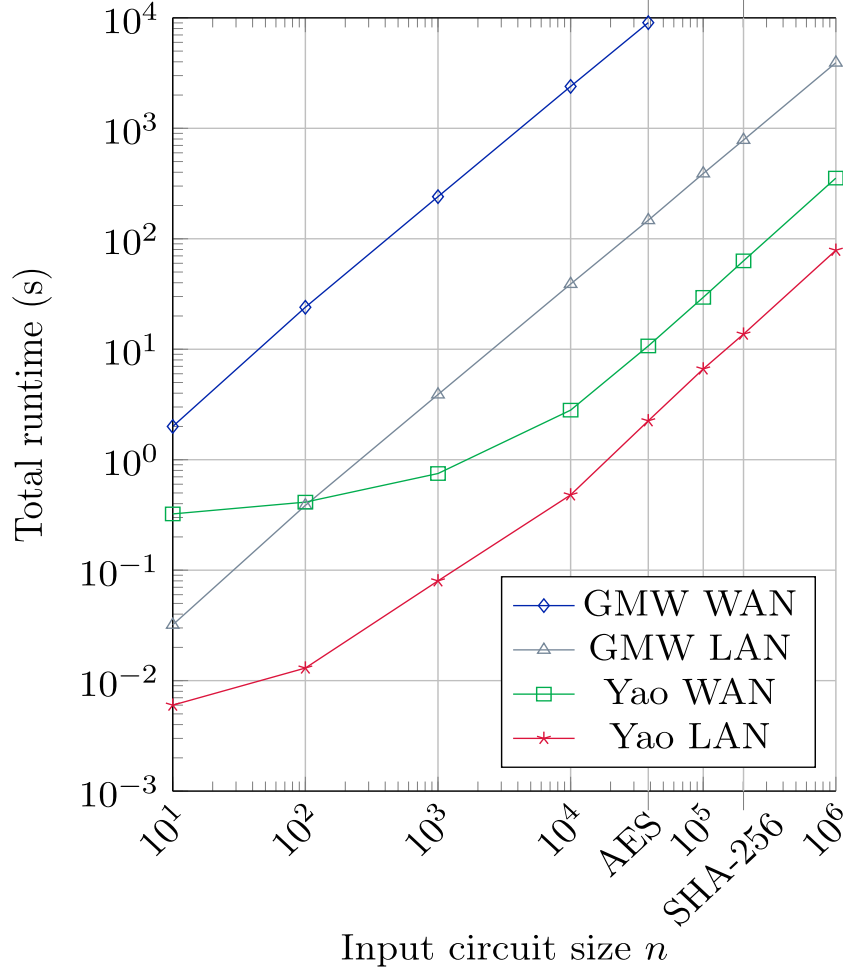
We include our implementation of these efficient UC building blocks in the open-source ABY framework <https://encrypto.de/code/ABY>. For evaluating a UC securely, the output universal circuit file of our UC compiler is parsed, a circuit  $UC$  is generated and evaluated with the input  $x$  and the control bits  $c^f$  to compute  $f(x)$ . Our toolchain is the first implementation of Valiant’s size-optimized UC that supports efficient private function evaluation [45].

## 8.2. Experimental Results

We validate the practicality of our implementation, which is the first practical implementation of private function evaluation (PFE), cf. Sect. 1.1. We ran our experiments on two Desktop PCs, each equipped with an Intel Core i9-7960X CPU with 2.8 GHz and 128 GB RAM. We give the runtimes in Fig. 15 and communication in Fig. 16 for our example circuits from the previous section, i.e., for random circuits of sizes  $n \in \{10, 100, \dots, 1,000,000\}$  as well as the AES and SHA-256 circuits from [65]. For completeness, we give the exact numbers in Table 7 in “Appendix D.” Our runtime measurements are provided from an average of 10 executions, in two different settings: in a LAN setting with 10 Gbit/s bandwidth and 1 ms RTT, as well as in a simulated WAN setting with 100 Mbit/s bandwidth and 100 ms RTT.

We evaluate UCs in ABY [19] with both the GMW protocol [32] and Yao’s garbled circuit protocol [69] with state-of-the-art optimizations. Yao’s garbled circuit protocol achieves much better runtimes than that of the GMW protocol since the latter has  $\mathcal{O}(n)$  rounds (i.e., the number of rounds is the depth of the circuit, and Valiant’s UCs have depth  $\mathcal{O}(n)$ , cf. Sect. 6.1 and Table 7 in “Appendix D”), whereas Yao’s protocol runs in 3 rounds. The effect of this is especially apparent in the WAN setting where the round-trip time is much higher. In both settings, the runtime of the GMW protocol is dominated by the linear term due to the linear number of online rounds. The amount of communication is similar in both implementations; however, it could be reduced by half for Yao’s protocol if X and Y switches would be implemented with the optimization from [43] using only one ciphertext. The current implementation utilizes two ciphertexts per X and Y switches.

Due to the clear advantage of Yao’s protocol over the GMW protocol, we highly recommend using Yao’s protocol when evaluating UCs securely for PFE. Investigating depth-optimized UCs [17] with  $\mathcal{O}(d)$  depth in the depth of the input circuit  $d$  could improve the performance of the GMW protocol; however, its number of rounds will still depend on  $d$ , whereas Yao’s protocol runs in only 3 rounds.

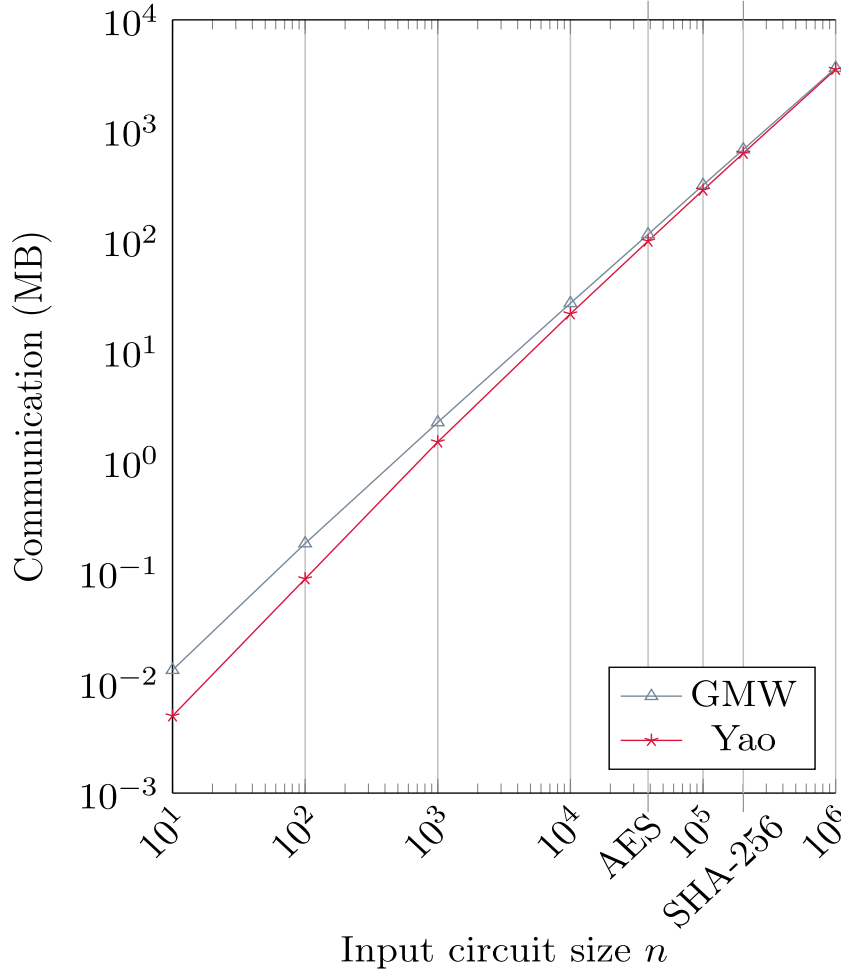


**Fig. 15.** Total runtime in seconds on LAN/WAN of PFE with the best available UC variant  $UC^H(\text{Valiant-2, Zhao et al.-4})$ .

### 8.3. Comparison of PFE Approaches

Mohassel et al. in [53] design a generic framework for PFE and apply it to three different scenarios: to the  $m$ -party GMW protocol [32], to Yao’s garbled circuits [70], and to arithmetic circuits using homomorphic encryption [16]. Both the two-party versions of their framework with the GMW protocol and the one with Yao’s garbled circuit protocol have two alternatives: Using homomorphic encryption, they achieve linear complexity  $\mathcal{O}(n)$  in the circuit size  $n$ , and when using a solution solely based on oblivious transfers (OTs), they obtain a construction with  $\mathcal{O}(n \log n)$  symmetric-key operations. The OT-based construction in both cases is more desirable in practice, since OT extension reduces the number of expensive public-key operations significantly [2, 36].

As the asymptotical complexity of this construction and using Valiant’s UC for PFE is the same, we compare these methods for PFE. We revisit the formulas provided in [53] for the PFE protocol based on Yao’s garbled circuits and elaborate on the number of symmetric-key operations when the different PFE protocols are used. Mohassel et al. show that the total number of switches in their framework is  $4\tilde{g} \log_2(2\tilde{g}) + 1$  that are evaluated using OT extension, for which they calculate  $8\tilde{g} \log_2(2\tilde{g}) + 8$  symmetric-key operations together with  $5\tilde{g}$  operations for evaluating the universal gates with Yao’s

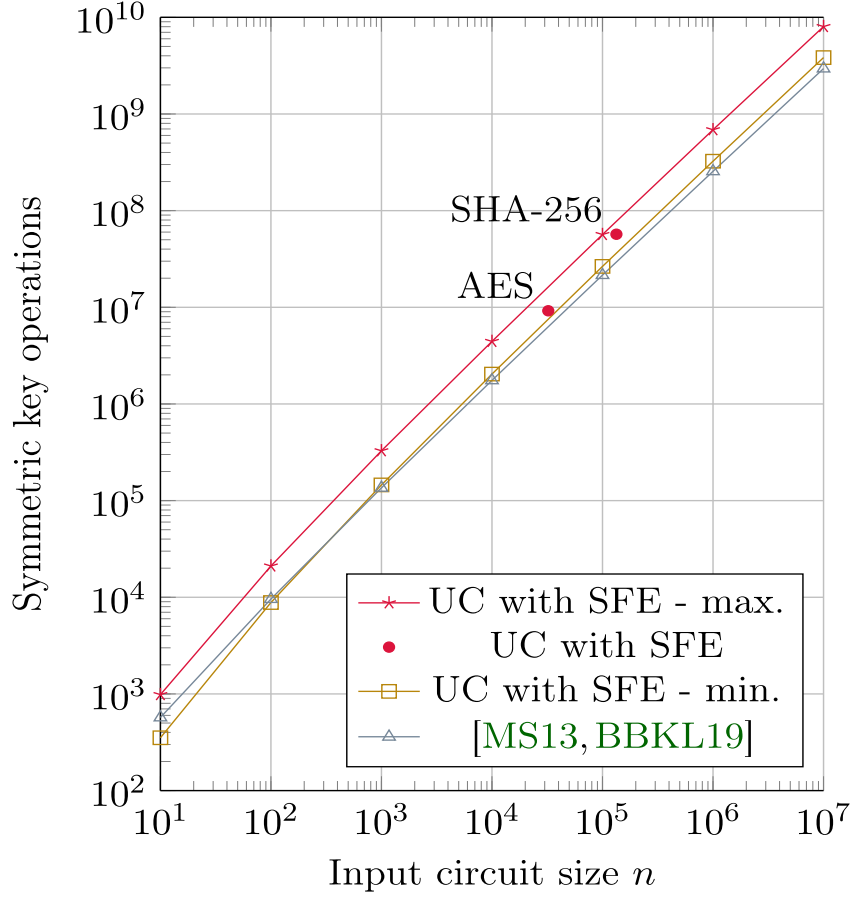


**Fig. 16.** Total communication in megabytes of PFE with the best available UC variant  $UC^H(\text{Valiant-2, Zhao et al.-4})$ .

protocol. We count only the work of the party that performs most of the work, i.e.,  $4\tilde{g}$  symmetric-key operations for creating a garbled circuit with  $\tilde{g}$  gates and 3 symmetric-key operations (two calls to a hash function and one call to a pseudorandom function (PRF)) for each OT using today's most efficient OT extension of [2]. Hence, according to our estimations, the protocol of [53] requires  $12\tilde{g} \log_2(2\tilde{g}) + 4\tilde{g} + 12$  symmetric-key operations.

In the same way, we assume that in the case of PFE with UCs, for both the universal gates and switches, the garbler needs  $4n$  symmetric-key operations. In this case, however,  $n = u + v + g$ , where  $\tilde{g} \leq g \leq 2\tilde{g} + v$ . It is, therefore, difficult to directly compare complexities of specifically designed protocols with  $\tilde{g}$  fanin-2 gates and UCs where the input circuit is required to have fanout 2 as well. In Fig. 17, we therefore depict the minimum and maximum required number of symmetric-key operations for circuits with size  $\tilde{g} \in \{10, 100, \dots, 1,000,000\}$ . Moreover, we depict the concrete values with real-world circuits (AES-128 and SHA-256 from [65]) with UC with SFE, and note that for the other approaches the points lie on the corresponding line.

The protocol of [53] has been improved to achieve better communication in [6]. The communication of the protocol of [53] is  $(10\tilde{g} \log_2 \tilde{g} + 4\tilde{g} + 5) \cdot 128$ , while that of



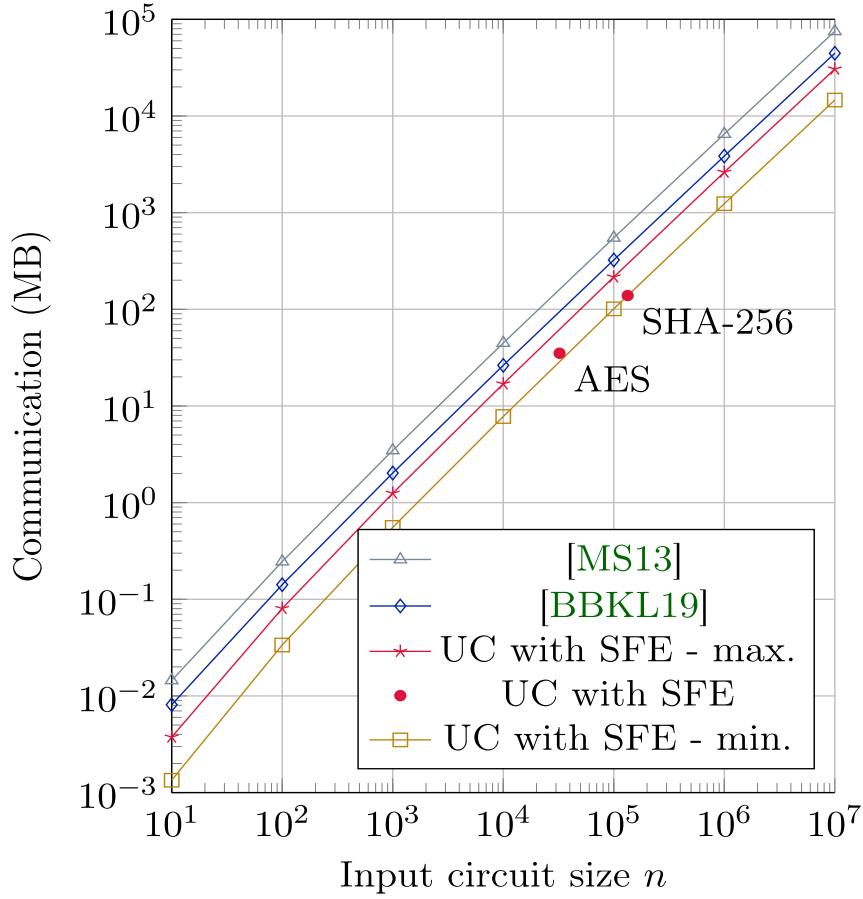
**Fig. 17.** The number of symmetric-key operations of different PFE protocols: Valiant’s UC with Yao’s garbled circuits, Mohassel et al.’s OT-based method from [53] and its optimized version from [6].

[6] is  $(6\tilde{g} \log_2 \tilde{g} + 0.5\tilde{g} + 3) \cdot 128$ . For SFE with UC, we require one ciphertext per  $X$  and  $Y$  switches [43] and  $3 \cdot 2$  ciphertexts per universal gates. Figure 18 depicts the comparison between the communication of SFE with UCs with minimum and maximum values depending on the relation of  $g$  and  $\tilde{g}$  as before and the alternatives of [53] and [6]. We can see that SFE with UCs always achieves the best communication, requiring  $1.5\text{--}3\times$  less communication than the improvement of [6].

## 9. Conclusion

Universal circuits (UCs) are highly relevant for various applications such as verifiable computation, attribute-based encryption, and private function evaluation (PFE) which can, for example, be used for privacy-preserving evaluation of diagnostic programs, proprietary software and in private database management systems. These applications require size-optimized universal circuits, first proposed by Valiant [66]. Since then, several optimizations appeared to further reduce the size of the UCs.

In this article, we revisit Valiant’s original constructions and the optimizations later proposed by our previous works by Kiss and Schneider [45] and Günther et al. [31] as well as by Zhao et al. [72]. We have shown the practicality of Valiant’s universal circuit



**Fig. 18.** Communication of different PFE protocols in megabytes:  $UC^H(\text{Valiant-2, Zhao et al.-4})$  with Yao’s garbled circuits, Mohassel et al.’s OT-based method from [53] and its optimized version from [6].

constructions and its several improvements by providing the implementation of the most efficient UC to date with size  $\sim 4.5n \log_2 n$  in the input circuit size  $n$ . Moreover, we highly improve the memory consumption of our UC generation algorithm by designing and implementing a method that utilizes  $\mathcal{O}(n)$  memory instead of the previous methods using  $\mathcal{O}(n \log n)$  memory.

Universal circuits for an input circuit size of one million can be generated and programmed within a matter of around 18 minutes on a standard PC and utilized in various applications. We demonstrate the practicality of PFE with the secure evaluation of UCs and show that such a large universal circuit can be evaluated within 1.3 and 5.9 minutes using Yao’s garbled circuit protocol in LAN and WAN settings, respectively.

### Acknowledgements

Open Access funding provided by Projekt DEAL. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program under Grant agreement No. 850990 PSOTI and by the European Union’s 7th Framework Program (FP7/2007-2013) under Grant agreement No. 609611 PRACTICE. It was co-funded by the Deutsche Forschungs-

gemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by the BMBF and HMWK within CRISP and ATHENE. We thank Michael Zohner for helping with the implementation in ABY and the anonymous reviewers of EUROCRYPT’16, ASIACRYPT’17, and JoC for their helpful comments.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

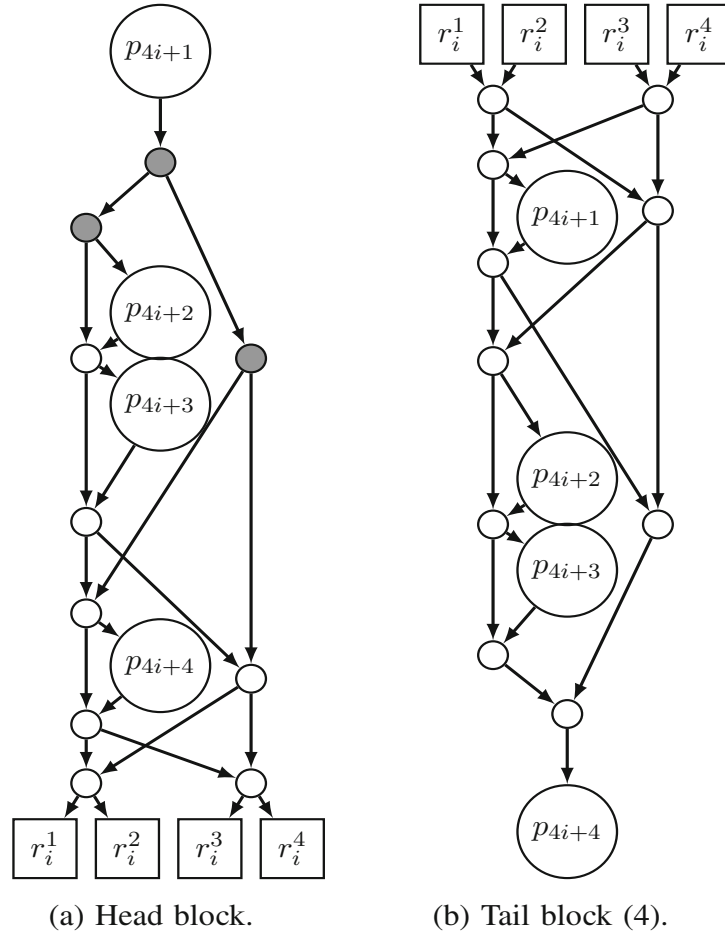
## A Abbreviations and Notations

ABE	Attribute-based encryption.
DAG	Directed acyclic graph.
DBMS	Database management system.
EUG	Edge-universal graph.
GRR3	Garbled row reduction.
OT	Oblivious transfer.
PFE	Private function evaluation.
semi-PFE	Semi-private function evaluation.
SFE	Secure function evaluation or secure two-party computation.
UC	Universal circuit.
$f$	Function to be privately evaluated using a universal circuit.
$c^f$	Control bits for a universal circuit to compute function $f$ .
$u$	Number of inputs in simulated Boolean circuit.
$v$	Number of outputs in simulated Boolean circuit.
$\hat{g}$	Number of gates in simulated Boolean circuit with arbitrary fanin and fanout.
$\tilde{g}$	Number of gates in simulated Boolean circuit with fanin 2 and arbitrary fanout.
$g$	Number of gates in simulated Boolean circuit with fanin and fanout 2.
$C_{u,v}^{\hat{g}}$	The Boolean circuit that describes $f$ with arbitrary fanin and fanout.
$C_{u,v}^{\tilde{g}}$	The Boolean circuit that describes $f$ with fanin 2 and arbitrary fanout.
$C_{u,v}^g$	The Boolean circuit that describes $f$ with fanin and fanout 2.
$n$	Size of the simulated circuit $C_{u,v}^g$ with fanin and fanout 2, $n = u + v + g$ .
$d$	Depth of the simulated circuit $C_{u,v}^g$ .
$G$	The $\Gamma_2(n)$ graph of $C_{u,v}^g$ where every input, output and gate is represented with a node and every wire is represented with an edge.
$\Gamma_\rho(n)$	The set of all graphs with fanin and fanout $\rho$ and $n$ nodes.
$U_n(\Gamma_\rho)$	Edge-universal graph for $\Gamma_\rho(n)$ graphs, used generally for Valiant’s UC.
$U_n^{(k)}(\Gamma_\rho)$	$k$ -way edge-universal graph for $\Gamma_\rho(n)$ graphs.
$U_n^{\text{hybrid}(K)}(\Gamma_\rho)$	Hybrid edge-universal graph for $\Gamma_\rho(n)$ graphs with a set $K$ of $k$ possible values, e.g., $K = \{2, 4\}$ .
$p_i$	Distinguished nodes in $U_n(\Gamma_\rho)$ , called poles, with fanin and fanout $\rho$ .
$P$	Set of all poles in $U_n(\Gamma_\rho)$ .
$U$	A universal gate that computes any function with two inputs and one output, using four control bits $c_0, c_1, c_2, c_3$ as in Eq. 1.
$X$	A two-output X-switching block that returns its two input values either in the same or in reversed order depending on control bit $c$ .
$Y$	A one-output Y-switching block that returns one of the two input values depending on control bit $c$ .

$B^{(k)}$	Body block of $k$ -way EUG.
$P^{(k)}$	Permutation network for $k$ nodes.
$P_1^{(k)}$	Lower bound on the size of the permutation network for $k$ nodes.
$P_W^{(k)}$	Size of the Waksman's permutation network [67] for $k$ nodes.
$U_n^{\text{KS08}}$	The UC of [44].
$\text{UC}^{\text{Valiant-2}}$	Valiant's 2-way UC [66].
$\text{UC}^{\text{Valiant-4}}$	Valiant's 4-way UC [66].
$\text{UC}^{\text{Zhao et al.-4}}$	Valiant's 4-way UC with Zhao et al.'s optimization [72].
$\text{UC}^{\text{H(Valiant-2,4)}}$	Hybrid UC with $\text{UC}^{\text{Valiant-2}}$ and $\text{UC}^{\text{Valiant-4}}$ .
$\text{UC}^{\text{H(Valiant-2, Zhao et al.-4)}}$	Hybrid UC with $\text{UC}^{\text{Valiant-2}}$ and $\text{UC}^{\text{Zhao et al.-4}}$ .

## B Optimized Blocks for Zhao et al.'s 4-way UC

In this section, we depict the head and tail block constructions in Fig. 19a and b, respectively, for Zhao et al.'s body block (cf. Fig. 5b), similar to those of [31, Figs. 4e-4f] for Valiant's 4-way UC. Similarly, tail blocks can be designed also for smaller number of poles in the final block, but as shown in Table 6, they will have the same size as our tail blocks for Valiant's 4-way UC [31, Fig. 4g-i].



**Fig. 19.** Optimized blocks for Zhao et al.'s 4-way block (Fig. 5b) [72].



## C Example Output of Our UC Compiler

In this section, we provide an example output of our UC compiler, i.e., the circuit and programming files shown on Listings 3 and 4 corresponding to the universal circuit shown in Fig. 1e on p. 10.

Listing 3. Example output UC.			Listing 4. Example programming.		
1	C 0 1	1	0	//input bits	
2	X 0 1 2	2	0	//X switch (no swap grey)	
3	X 1 0 3	3	1	//X switch (swap green)	
4	X 0 2 4	4	1	//X switch (swap blue)	
5	X 3 0 5	5	0	//X switch (undefined)	
6	U 2 3 6	6	1	//A N D gate (0001)	
7	X 4 6 7	7	1	//X switch (swap blue)	
8	X 6 5 8	8	0	//X switch (no swap red)	
9	Y 4 7 9	9	0	//Y switch (undefined)	
10	Y 8 5 10	10	0	//Y switch (undefined)	
11	U 7 8 11	11	6	//X O R gate (0110)	
12	Y 9 11 12	12	0	//Y switch (right input orange)	
13	Y 11 10 13	13	0	//Y switch (undefined)	
14	Y 12 13 14	14	1	//Y switch (left input orange)	
15	O 14	15		//output bits	

## D Concrete Performance Measures for Private Function Evaluation

In this section, we provide the concrete performance measures used for depicting the runtimes and communication of PFE by securely evaluating UCs generated with  $UC^H(\text{Valiant-2, Zhao et al.-4})$  in Figs. 15 and 16, respectively.

**Table 7.** Runtime and communication of PFE with universal circuits generated for input circuit size  $n$  (cf. Table 2 for the respective UC sizes).

Input circuit size $n$	10	100	1000	10,000	38,518 (AES)	100,000	201,206 (SHA-256)	1,000,000
Yao LAN (s)	0.006	0.013	0.08	0.48	2.25	6.63	13.70	78.73
GMW LAN (s)	0.032	0.386	3.89	38.92	147.14	389.85	783.94	3 925.94
Yao WAN (s)	0.323	0.413	0.75	2.81	10.71	29.49	63.10	354.32
GMW WAN (s)	2.000	23.990	240.55	2395.32	9044.30	*	*	*
Yao comm. (MB)	0.005	0.087	1.51	21.79	99.36	287.51	620.07	3 562.21
GMW comm. (MB)	0.013	0.182	2.27	27.20	114.21	319.31	670.23	3660.17
GMW rounds	34	441	4491	44,991	169,871	449,991	903,686	4,499,991

\*Denotes cases where an experiment would have taken more than 5 hours and therefore was not performed

## References

- [1] M. Abadi, J. Feigenbaum, Secure circuit evaluation. *J. Cryptology*. 2(1), 1–12 (1990)
- [2] G. Asharov, Y. Lindell, T. Schneider, M. Zohner, More efficient oblivious transfer and extensions for faster secure computation, in *CCS' 13*. (ACM, 2013), pp. 535–548
- [3] A. Afshar, P. Mohassel, B. Pinkas, B. Riva, Non-interactive secure computation based on cut-and-choose, in *EUROCRYPT'14*. LNCS, vol. 8441 (Springer, 2014), pp. 387–404
- [4] N. Attrapadung, Fully secure and succinct attribute based encryption for circuits from multi-linear maps. *Cryptology ePrint Archive, Report 2014/772*, (2014). <https://ia.cr/2014/772>

- [5] O. Bıçer, M. A. Bingöl, M. S. Kiraz, Highly efficient and reusable private function evaluation with linear complexity. *Cryptology ePrint Archive, Report 2018/515*, (2018) <https://ia.cr/2018/515>
- [6] M. A. Bingöl, O. Bıçer, M. S. Kiraz, A. Levi, An efficient 2-party private function evaluation protocol based on half gates. *Comput. J.* **62**(4), 598–613
- [7] B. Beauquier, É. Darrot, On arbitrary size Waksman networks and their vulnerability. *Parallel Process. Lett.* **12**(3-4), 287–296 (2002)
- [8] D. Bera, S. A. Fenner, F. Green, S. Homer, Efficient universal quantum circuits. *Quantum Inf. Comput.* **10**(1–2), 16–27 (2010)
- [9] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi, T. Schneider, Secure evaluation of private linear branching programs with medical applications, in *ESORICS'09*. LNCS, vol. 5789 (Springer, 2009), pp. 424–439
- [10] N. Büscher, S. Katzenbeisser, *Compilation for Secure Multi-party Computation*. Springer Briefs in Computer Science. (Springer, 2017)
- [11] A. Ben-David, N. Nisan, B. Pinkas, FairplayMP: A system for secure multi-party computation, in *CCS'08*. (ACM, 2008), pp. 257–266
- [12] S. Banescu, M. Ochoa, N. Kunze, A. Pretschner, Idea: Benchmarking indistinguishability obfuscation - A candidate implementation, in *Engineering Secure Software and Systems (ESSoS'15)*. LNCS, vol. 8978 (Springer, 2015), pp. 149–156
- [13] J. Brickell, D. E. Porter, V. Shmatikov, E. Witchel, Privacy-preserving remote diagnostics, in *CCS'07*. (ACM, 2007), pp. 498–507
- [14] N. Bitansky, V. Vaikuntanathan, Indistinguishability obfuscation from functional encryption, in *FOCS'15*. (IEEE, 2015), pp. 171–190
- [15] C. Cachin, J. Camenisch, J. Kilian, J. Müller, One-round secure computation and secure autonomous mobile agents, in *ICALP'00*. LNCS, vol. 1853 (Springer, 2000), pp. 512–523
- [16] R. Cramer, I. Damgård, J. B. Nielsen, Multiparty computation from threshold homomorphic encryption, in *EUROCRYPT'01*. LNCS, vol. 2045 (Springer, 2001), pp. 280–299
- [17] S. A. Cook, H. J. Hoover, A depth-universal circuit. *SIAM J. Computing.* **14**(4), 833–839 (1985)
- [18] K. Durnoga, S. Dziembowski, T. Kazana, M. Zajac, One-time programs with limited memory, in *Information Security and Cryptology (INSCRYPT'13)*. LNCS, vol. 8567 (Springer, 2013), pp. 377–394
- [19] D. Demmler, T. Schneider, M. Zohner, ABY—a framework for efficient mixed-protocol secure two-party computation, in *NDSS'15*. (The Internet Society, 2015). Code: <https://crypto.de/code/ABY>
- [20] K. B. Frikken, M. J. Atallah, J. Li, Attribute-based access control with hidden policies and hidden credentials. *IEEE Trans. Comput.* **55**(10), 1259–1270, (2006).
- [21] K. B. Frikken, M. J. Atallah, C. Zhang, Privacy-preserving credit checking, in *Electronic Commerce (EC'05)*. (ACM, 2005), pp. 147–154
- [22] D. Fiore, R. Gennaro, V. Pastro, Efficiently verifiable computation on encrypted data, in *CCS'15*. (ACM, 2014), pp. 844–855
- [23] M. Franz, A. Holzer, S. Katzenbeisser, C. Schallhart, H. Veith, CBMC-GC: an ANSI C compiler for secure two-party computations, in *Conference on Compiler Construction (CC'14)*. LNCS, vol. 8409 (Springer, 2014), pp. 244–249
- [24] K. B. Frikken, J. Li, M. J. Atallah, Trust negotiation with hidden credentials, hidden policies, and policy cycles, in *NDSS'06*. (The Internet Society, 2006), pp. 157–172
- [25] B. Fisch, B. Vo, F. Krell, A. Kumarasubramanian, V. Kolesnikov, T. Malkin, S. M. Bellovin, Malicious-client security in Blind Seer: A scalable private DBMS. in *IEEE S&P'15*. (IEEE, 2015), pp. 395–410
- [26] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, B. Waters, Candidate indistinguishability obfuscation and functional encryption for all circuits, in *FOCS'13*. (IEEE, 2013), pp. 40–49
- [27] S. Garg, C. Gentry, S. Halevi, A. Sahai, B. Waters, Attribute-based encryption for circuits from multilinear maps, in *CRYPTO'13*. LNCS, vol. 8043 (Springer, 2013), pp. 479–499
- [28] S. Garg, C. Gentry, S. Halevi, M. Zhandry, Fully secure attribute based encryption from multilinear maps. *Cryptology ePrint Archive, Report 2014/622*, (2014) <https://ia.cr/2014/622>
- [29] R. Gennaro, C. Gentry, B. Parno, M. Raykova, Quadratic span programs and succinct NIZKs without PCPs, in *EUROCRYPT'13*, volume 7881 of LNCS, (Springer, 2013), pp. 626–645
- [30] C. Gentry, S. Halevi, V. Vaikuntanathan, i-hop homomorphic encryption and rerandomizable Yao circuits, in *CRYPTO'10*. LNCS, vol. 6223 (Springer, 2010), pp. 155–172

- [31] D. Günther, Á. Kiss, T. Schneider, More efficient universal circuit constructions, in *ASIACRYPT'17*. LNCS, vol. 10625. (Springer, 2017), pp. 443–470. Full version: <https://ia.cr/2017/798>, Code: <https://crypto.de/code/UC>
- [32] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or a completeness theorem for protocols with honest majority, in *STOC'87*. (ACM, 1987), pp. 218–229
- [33] Z. Galil, W. J. Paul, An efficient general purpose parallel computer, in *STOC'81*. (ACM, 1981), pp. 247–262
- [34] S. Gorbunov, V. Vaikuntanathan, H. Wee, Attribute-based encryption for circuits, in: *STOC'13*. (ACM, 2013), pp. 545–554
- [35] Y. Huang, J. Katz, V. Kolesnikov, R. Kumaresan, A. J. Malozemoff, Amortizing garbled circuits, in *CRYPTO'14*. LNCS, vol. 8617 (Springer, 2014), pp. 458–475
- [36] Y. Ishai, J. Kilian, K. Nissim, E. Petrank, Extending oblivious transfers efficiently, in *CRYPTO'03*. LNCS, vol. 2729 (Springer, 2003), pp. 145–161
- [37] Y. Ishai, A. Paskin, Evaluating branching programs on encrypted data, in: *TCC'07*. LNCS, vol. 4392 (Springer, 2007), pp. 575–594
- [38] D. Kőnig. Gráfok és mátrixok, in *Matematikai és Fizikai Lapok*, vol. 38, pp. 116–119, (1931)
- [39] W. S. Kennedy, V. Kolesnikov, G. T. Wilfong, Overlaying conditional circuit clauses for secure computation, in *ASIACRYPT'17*. LNCS, vol. 10625 (Springer, 2017), pp. 499–528
- [40] J. Katz, L. Malka, Constant-round private function evaluation with linear complexity, in *ASIACRYPT'11*. LNCS, vol. 7073 (Springer, 2011), pp. 556–571
- [41] V. Kolesnikov, Free IF: How to omit inactive branches and implement  $S$ -universal garbled circuit (almost) for free, in *ASIACRYPT'18*. LNCS, vol. 11274 (Springer, 2018), pp. 34–58
- [42] S. Kamara, M. Raykova, Secure outsourced computation in a multi-tenant cloud, in *IBM Workshop on Cryptography and Security in Clouds, Zürich, Switzerland*, (2011)
- [43] V. Kolesnikov, T. Schneider, Improved garbled circuit: Free XOR gates and applications, in *ICALP'08*. LNCS, vol. 5126 (Springer, 2008), pp. 486–498
- [44] V. Kolesnikov, T. Schneider, A practical universal circuit construction and secure evaluation of private functions, in *FC'08*. LNCS, vol. 5143 (Springer, 2008), pp. 83–97. Code: <https://crypto.de/code/FairplayPF>
- [45] Á. Kiss, T. Schneider, Valiant's universal circuit is practical, in *EUROCRYPT'16*. LNCS, vol. 9665 (Springer, 2016), pp. 699–728. Full version: <https://ia.cr/2016/093>, Code: <https://crypto.de/code/UC>
- [46] H. Lipmaa, P. Mohassel, S. S. Sadeghian, Valiant's universal circuit: Improvements, implementation, and applications. *Cryptology ePrint Archive, Report 2016/017*, (2016). <https://ia.cr/2016/017>
- [47] Y. Lindell, B. Pinkas, A proof of security of Yao's protocol for two-party computation. *J. Cryptology*. **22**(2), 161–188 (2009)
- [48] L. Lovász, M. D. Plummer, *Matching Theory*. AMS Chelsea Publishing Series. (American Mathematical Society, Providence, 2009)
- [49] Y. Lindell, B. Riva, Blazing fast 2PC in the offline/online setting with security for malicious adversaries, in *CCS'15*. (ACM, 2015), pp. 579–590
- [50] F. Meyer auf der Heide, Efficiency of universal parallel computers, in *Theoretical Computer Science*. LNCS, vol. 145 (Springer, 1983), pp. 221–241
- [51] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, Fairplay—A secure two-party computation system, in *USENIX Security'04*. (USENIX, 2004), pp. 287–302
- [52] P. Mohassel, M. Rosulek, Non-interactive secure 2PC in the offline/online and batch settings, in *EUROCRYPT'17*. LNCS, vol. 10212 (Springer, 2017), pp. 425–455
- [53] P. Mohassel, S. S. Sadeghian, How to hide circuits in MPC—An efficient framework for private function evaluation, in *EUROCRYPT'13*. LNCS, vol. 7881 (Springer, 2013), pp. 557–574
- [54] P. Mohassel, S. S. Sadeghian, N. P. Smart, Actively secure private function evaluation, in *ASIACRYPT'14*. LNCS, vol. 8874 (Springer, 2014), pp. 486–505
- [55] M. Naor, B. Pinkas, R. Sumner, Privacy preserving auctions and mechanism design, in *Electronic Commerce (EC'99)*. (1999), pp. 129–139
- [56] S. Niksefat, B. Sadeghiyan, P. Mohassel, S. S. Sadeghian, ZIDS: A privacy-preserving intrusion detection system using secure two-party computation protocols. *Comput. J.* **57**(4), 494–509 (2014)
- [57] R. Ostrovsky, W. E. Skeith III, Private searching on streaming data, in *CRYPTO'05*. LNCS, vol. 3621 (Springer, 2005), pp. 223–240

- [58] B. Pinkas, Cryptographic techniques for privacy-preserving data mining. *SIGKDD Explor.* **4**(2), 12–19 (2002)
- [59] V. Pappas, F. Krell, B. Vo, V. Kolesnikov, T. Malkin, S. Geol Choi, W. George, A. D. Keromytis, S. Bellovin, Blind Seer: A scalable private DBMS, in *IEEE S&P' 14*. (IEEE, 2014), pp. 359–374
- [60] A. Paus, A.-R. Sadeghi, T. Schneider, Practical secure evaluation of semi-private functions, in *ACNS'09*. LNCS, vol. 5536 (Springer, 2009), pp. 89–106
- [61] T. Schneider, Practical secure function evaluation. Master's thesis, University Erlangen-Nürnberg, Germany, (2008)
- [62] C. Shannon, The synthesis of two-terminal switching circuits. *Bell Labs Tech. J.* **28**(1), 59–98 (1949)
- [63] T. Sander, A. L. Young, M. Yung, Non-interactive cryptocomputing for  $NC^1$ , in *FOCS' 99*. (IEEE, 1999), pp. 554–567
- [64] T. Schneider, M. Zohner, GMW vs. Yao? Efficient secure two-party computation with low depth circuits, in *Financial Cryptography and Data Security (FC'13)*. LNCS, vol. 7859 (Springer, 2013), pp. 275–292
- [65] S. Tillich, N. Smart, Circuits of basic functions suitable for MPC and FHE, (2018) <https://homes.esat.kuleuven.be/~nsmart/MPC/>
- [66] L. G. Valiant, Universal circuits (preliminary report), in *STOC' 76*. (ACM, 1976), pp. 196–203
- [67] A. Waksman, A permutation network. *J. ACM.* **15**(1), 159–163 (1968)
- [68] I. Wegener, *The complexity of Boolean functions*. Wiley-Teubner, Hoboken (1987)
- [69] A. C.-C. Yao, Protocols for secure computations (extended abstract), in *FOCS' 82*. (IEEE, 1982), pp. 160–164
- [70] A. C.-C. Yao, How to generate and exchange secrets (extended abstract), in *FOCS' 86*. (IEEE, 1986), pp. 162–167
- [71] J. Zimmerman, How to obfuscate programs directly, in *EUROCRYPT'15*. LNCS, vol. 9057 (Springer, 2015), pp. 439–467
- [72] S. Zhao, Y. Yu, J. Zhang, H. Liu, Valiant's universal circuits revisited: an overall improvement and a lower bound, in *ASIACRYPT'19*. LNCS, vol. 11921 (Springer, 2019), pp. 401–425

## D Linear-Complexity Private Function Evaluation is Practical (ESORICS'20)

---

- [HKRS20] M. HOLZ, Á. KISS, D. RATHEE, T. SCHNEIDER. “**Linear-Complexity Private Function Evaluation is Practical**”. In: *25. European Symposium on Research in Computer Security (ESORICS'20)*. Vol. 12309. LNCS. Full version: <https://ia.cr/2020/853>. Code: <https://encrypto.de/code/linearPFE>. Springer, 2020, pp. 401–420. CORE Rank A. Appendix D.

[http://doi.org/10.1007/978-3-030-59013-0\\_20](http://doi.org/10.1007/978-3-030-59013-0_20)



# Linear-Complexity Private Function Evaluation is Practical

Marco Holz<sup>1</sup>(✉), Ágnes Kiss<sup>1</sup>, Deevashwer Rathee<sup>2</sup>, and Thomas Schneider<sup>1</sup>

<sup>1</sup> ENCRYPTO, Technische Universität Darmstadt, Darmstadt, Germany  
`{holz,kiss,schneider}@encrypto.cs.tu-darmstadt.de`

<sup>2</sup> Department of Computer Science, IIT (BHU) Varanasi, Varanasi, India  
`deevashwer.student.cse15@iitbhu.ac.in`

**Abstract.** Private function evaluation (PFE) allows to obliviously evaluate a private function on private inputs. PFE has several applications such as privacy-preserving credit checking and user-specific insurance tariffs. Recently, PFE protocols based on universal circuits (UCs), that have an inevitable superlinear overhead, have been investigated thoroughly. Specialized public key-based protocols with linear complexity were believed to be less efficient than UC-based approaches.

In this paper, we take another look at the linear-complexity PFE protocol by Katz and Malka (ASIACRYPT'11): We propose several optimizations and split the protocol in different phases that depend on the function and inputs respectively. We show that HE-based PFE is practical when instantiated with state-of-the-art ECC and RLWE-based homomorphic encryption. Our most efficient implementation outperforms the most recent UC-based PFE implementation of Alhassan et al. (JoC'20) in communication for all circuit sizes and in computation starting from circuits of a few thousand gates already.

**Keywords:** Private function evaluation · Homomorphic encryption · Secure computation

## 1 Introduction

While computations on a local machine can be secured against malicious eavesdropping, computations that are performed collaboratively on two or more devices typically rely on the trustworthiness of remote systems. This poses a risk to the sensitive data supplied by the participants. Privacy-preserving protocols aim to mitigate these risks by protecting the data using cryptographic approaches such that there is no need for a trusted remote party anymore.

Secure two-party computation (STPC) or secure function evaluation (SFE) protocols allow two parties to jointly compute a function on private data without learning the other party's inputs. Private function evaluation (PFE) extends this setting by also hiding the evaluated function from one of the parties:  $P_1$  inputs a private function  $f$ , typically represented by a circuit  $\mathcal{C}_f$ , and  $P_2$  inputs private data  $x$  and learns only  $f(x)$  but no additional information on  $f$  (except its size).



PFE has diverse applications that require to keep the participants' inputs private and hide the operations applied to these inputs from one of the participants. We describe a few example applications. In a *privacy-preserving intrusion detection system (IDS)* [Nik+14], a server holds a set of zero-day signatures (including regular expressions matching the payload) and is able to check whether sensitive data uploaded to the IDS matches those signatures such that the server learns nothing about the data and the client learns nothing about the signatures. Using PFE, *attribute-based access control* can be enhanced to protect both sensitive credentials and sensitive policies [FAL06]. PFE can be used for *privacy-preserving credit worthiness checking* [FAZ05], disclosing neither the customer's private financial data nor the private criteria of the loaner. In *privacy-preserving car insurance rate calculation* [Gün+19] the privacy-critical customer data, as well as the tariff calculation details remain private.

The most common approach for PFE is to reduce it to classical SFE by securely evaluating a public universal circuit (UC) [Val76, KS08a, KS16, LMS16, GKS17, Alh+20, Zha+19, Liu+20]. This series of works on optimizations and implementations of UCs has shown that UC-based PFE can be practical, but UCs introduce an inevitable logarithmic overhead [Val76]. Katz and Malka [KM11] propose a linear-complexity PFE scheme based on homomorphic encryption (HE) and Yao's garbled circuit protocol. They expect their scheme to be "easier to implement and more efficient (for larger circuits) than approaches relying on universal circuits". However, their scheme has not been implemented yet.

**Our Contributions.** Our paper takes another look at the linear-complexity PFE protocol by Katz and Malka [KM11]. We split the protocol into several phases so that parts of the protocol can be precomputed knowing, e.g., only the size of the private function or the private function itself. For instance, for a privacy-preserving IDS it is reasonable to precompute any function-dependent part so that the online phase where the client provides its input is fast. We optimize, instantiate, and implement their scheme using three state-of-the-art homomorphic encryption (HE) schemes: Elliptic curve (EC) ElGamal [Elg85], the Brakerski/Fan-Vercauteren (BFV) scheme [FV12], and the cryptosystem by Damgård/Jurik/Nielsen (DJN) [DJN10]. We implement our protocols using the ABY framework [DSZ15] and thereby provide the first implementation of a linear-complexity PFE scheme. Our experiments show that HE-based PFE outperforms today's most efficient UC-based PFE implementation [Alh+20] on the same platform already starting from circuits with only a few thousand gates.

## 2 Related Work

In this paper, we focus on PFE protocols that provide security against semi-honest adversaries. These can be categorized as follows:

**UC-Based PFE.** A universal circuit (UC) is a circuit that can be programmed to evaluate any Boolean circuit up to size  $n$  by specifying a set of program bits as its input. In recent years, a lot of research was put into optimizing and



implementing UC-based PFE, which reduces the task of PFE to standard SFE that relies mostly on symmetric cryptography where the function is the publicly known UC. Valiant [Val76] proposed two recursive UCs with sizes  $\sim 5n \log_2 n$  and  $\sim 4.75n \log_2 n$  in the size of the simulated circuit  $n$ , which are optimal up to a constant factor because any UC must have size at least  $\Omega(n \log n)$ . Zhao et al. [Zha+19] present a UC with size  $\sim 4.5n \log_2 n$ . A hybrid UC with size  $\sim 4.5n \log_2 n$ , combining optimizations from [KS16, GKS17, Zha+19] was implemented in [Alh+20]. The most recent UC from [Liu+20] has size  $\sim 3n \log_2 n$ . These constructions have reached lower bounds for the most common ways UCs are constructed [Zha+19, Liu+20], so no significant improvements are expected.

**OT-Based PFE.** Mohassel and Sadeghian introduce an oblivious transfer (OT)-based approach based on the oblivious evaluation of a switching network of size  $\Theta(n \log n)$  that hides the topology of the Boolean circuit [MS13]. Bingöl et al. [Bin+18] adapt the half gates optimization [ZRE15] to the OT-based approach of [MS13] and reduce the number of OTs by half. As shown in [Alh+20], the communication of both [MS13] and [Bin+18] is worse than that of UC-based PFE. PFE schemes based on both UCs and switching networks have an inevitable logarithmic overhead.

**TEE-Based PFE.** Felsen et al. [Fel+19] propose private function evaluation with a different trust assumption and implement PFE using Intel SGX as trusted execution environment (TEE), by evaluating a UC within the SGX enclave.

**HE-Based PFE.** The protocol by Katz and Malka [KM11] has linear complexity  $\mathcal{O}(n)$ , but its concrete practicality has not yet been explored. The authors use homomorphic encryption to hide the topology of the circuit  $\mathcal{C}_f$  from the party that obviously garbles the circuit (cf. Sect. 4.1). Mohassel and Sadeghian [MS13] include a linear-complexity protocol in their generic framework for PFE. They optimize the baseline protocol of [KM11], but their protocol is not more efficient than the improved protocol of [KM11] which we use. Mohassel et al. [MSS14] extend the protocol from [KM11, MS13] to security against malicious adversaries using zero-knowledge proofs while maintaining linear complexity. Biçer et al. present a reusable linear-complexity PFE scheme [Biç+18] based on the protocol of [KM11] which is efficient if the same private function  $f$  is evaluated multiple times. Their protocol in the first execution has slightly lower total communication, but around a factor four higher online computation than [KM11] (cf. [Biç+18, Table 1]). Later runs of the protocol with the same function are more efficient both in communication and computation than [KM11]. We leave investigating the concrete efficiency of the protocol of [Biç+18] for applications where the same function can be reused as future work.

In our paper, we resurrect the neglected line of research on linear-complexity HE-based PFE protocols and show that the protocol of [KM11] is practical.

### 3 Preliminaries

In this section, we describe preliminaries to our work from the fields of secure function evaluation (SFE) in Sect. 3.1 and private function evaluation (PFE) in Sect. 3.2, and recapitulate the homomorphic encryption (HE) schemes we use in Sect. 3.3.

#### 3.1 Circuit-Based Secure Function Evaluation

We focus on security against semi-honest (passive) adversaries where all parties are assumed to follow the protocol. This allows for highly efficient protocols and is a starting point for constructing protocols with stronger security guarantees.

In the past, several SFE protocols have been proposed that rely on a circuit representation of the function  $f$  which is known to both parties, e.g., Yao’s garbled circuit (GC) protocol [Yao82, Yao86, LP09] and the GMW protocol [GMW87]. In Yao’s protocol, party  $P_1$ , *the garbler*, prepares an encrypted version of the circuit in the form of garbled tables, which are then sent to  $P_2$ . The other party  $P_2$ , *the evaluator*, evaluates the *garbled circuit* after receiving the keys corresponding to his input wires using oblivious transfers.<sup>1</sup> Oblivious transfer (OT) allows the receiver  $P_2$  to retrieve one of two messages obliviously from the sender  $P_1$  without the receiver learning the other message or the sender learning which message was retrieved. Though OTs require expensive public-key cryptography [IR89], OT extension [Ish+03, Ash+13] allows to perform a large number of OTs more efficiently by extending a few *base OTs* and obtain many oblivious transfers using only symmetric cryptographic operations. Recent optimizations to Yao’s GC protocol include *point-and-permute* [BMR90], *free-XOR* [KS08b], *fixed-key AES garbling* [Bel+13], and *half gates* [ZRE15].

#### 3.2 Private Function Evaluation

Private function evaluation (PFE) extends SFE to the case where only one party  $P_1$  inputs a private function  $f$  represented by circuit  $\mathcal{C}_f$ . The protocol must guarantee that  $P_2$  on private input  $x$  learns the output  $f(x)$  but no other information about the function  $f$  whereas  $P_1$  learns nothing.<sup>2</sup> Generally, PFE protocols reveal the size of the circuit  $\mathcal{C}_f$  to the participants. If needed, the actual number of gates and wires can be hidden by adding dummy gates and dummy input/output wires to the circuit. One notable characteristic of PFE protocols is that  $P_1$  typically must not be able to learn the output of the function  $f$ . The reason for this is that an adversarial party  $P_1$  could reveal the inputs of party  $P_2$  by defining  $f$  to leak information about  $x$ , e.g.,  $f(x) = x$ .

<sup>1</sup> Even though the gates are encrypted and thus the gates’ types can easily be hidden from  $P_2$ ,  $P_2$  must know the *topology* of the circuit for evaluating the garbled circuit.

<sup>2</sup> This can be extended to the case where  $P_1$  also holds an input value in addition to the circuit  $\mathcal{C}_f$ . Our 2-party PFE implementation supports input values for both parties.

### 3.3 Homomorphic Encryption

Homomorphic encryption (HE) schemes allow for computations on encrypted data, i.e., operations performed on the ciphertexts are reflected in the output of decryption as if they were applied directly on the plaintexts.

The protocol of Katz and Malka [KM11] is based on additively homomorphic encryption, i.e., a HE scheme that supports only homomorphic addition. The authors of [KM11] suggest to instantiate their protocol with Paillier [Pai99] or ElGamal [Elg85] HE and mention that their protocol can be improved by using elliptic-curve cryptography (ECC). Since then, several significant improvements on additively HE were published that we consider in our implementation:

**DJN.** The DJN cryptosystem [DJN10], a generalization of Paillier’s scheme [Pai99], has since then been optimized using CRT-based decryption [HMS12]. Our implementation is based on *libpailler*<sup>3</sup> and uses this optimization.

**EC ElGamal.** EC ElGamal encryption offers exceptionally small ciphertexts, practical computation and an additive homomorphism over the underlying elliptic curve group. The use of elliptic curves over finite fields as a basis for a cryptosystem was suggested independently from each other by both Koblitz [Kob87] and Miller [Mil86]. In our implementation, we use the *RELIC Toolkit* [AG09] for ECC.

**BFV.** Significant improvements have been made in the area of RLWE-based HE [Reg05, LPR10, Bra12, FV12]. The RLWE-based BFV scheme [FV12, Lai17] is implemented in the Microsoft SEAL library [Sea19], which is among the fastest HE libraries available today. We present a high level overview of the BFV scheme restricted to only the part of its functionality which is relevant for our application. For additional details, see [Lai17]. We note that our discussion also applies to other popular Ring-LWE-based HE schemes such as BGV [BGV12].

The BFV scheme operates on polynomial rings of the form  $R = \mathbb{Z}[x]/(x^n + 1)$ , where the *polynomial modulus degree*  $n$  is a power of 2. For a *plaintext modulus*  $t$ , the plaintext space is defined as  $R_t = R/tR = \mathbb{Z}_t[x]/(x^n + 1)$ , which consists of polynomials of degree  $n - 1$  with coefficients in  $\mathbb{Z}_t$ . Similarly, the ciphertext space is defined as  $(R_q)^2$ , where  $q$  is called the *coefficient modulus* and  $R_q = R/qR$ . The encryption function  $\text{Enc}$  is probabilistic, takes a public key  $pk$  and a message  $m \in R_t$  as inputs, and outputs a ciphertext  $c \in (R_q)^2$ . The ciphertext output by  $\text{Enc}$  has a noise component associated with it which is necessary for maintaining security. The decryption function  $\text{Dec}$  takes the secret key  $sk$  and a ciphertext  $c \in (R_q)^2$  as inputs, and outputs a message  $m \in R_t$ . Decryption  $m = \text{Dec}(sk, \text{Enc}(pk, m))$  works if the ciphertext noise is below a certain threshold defined by the scheme parameters. For ease of exposition, we omit the keys from the invocation of the encryption and decryption functions, and assume a single key-pair throughout the paper, which makes the functions compatible.

$\text{Enc}$  is a homomorphic map from  $(R_t, +)$  to  $((R_q)^2, +)$ , which provides the scheme with its additive homomorphic properties. Given ciphertexts

<sup>3</sup> <http://hms.isi.jhu.edu/acsc/libpailler/>

$c_1 = \text{Enc}(m_1)$  and  $c_2 = \text{Enc}(m_2)$ , we have  $\text{Dec}(c_1 + c_2) = \text{Dec}(c_1) + \text{Dec}(c_2)$ . The noise component grows as we perform homomorphic operations on the ciphertext until it reaches a threshold, beyond which decryption is not possible and the ciphertext is rendered useless. This is not a problem since addition does not grow the noise by much. The scheme described so far only provides IND-CPA security against parties other than the key owner. To hide the operations applied to the ciphertext from the key owner, which may include some private inputs from other parties, and only reveal the result of decryption, the ciphertext needs to be flooded with extra noise (cf. [Lai17], § 9.4). This requires larger parameters to accommodate the extra noise, and has been taken into account in our parameter selection.

## 4 Linear-Complexity Private Function Evaluation

In this section, we recapitulate the private function evaluation (PFE) protocol of Katz and Malka [KM11] in Sect. 4.1, introduce further improvements in Sect. 4.2, and propose efficient instantiations using EC ElGamal in Sect. 4.3 and the BFV homomorphic encryption scheme in Sect. 4.4.

### 4.1 The [KM11] Protocol

The PFE protocol proposed by Katz and Malka [KM11] combines homomorphic encryption (HE) with Yao’s garbled circuit (GC) protocol to hide the topology of the circuit  $\mathcal{C}_f$  in addition to the parties’ inputs. They give a baseline protocol and a roughly twice as efficient improved protocol. We describe the improved protocol shown in Fig. 1 and refer to the original paper for the baseline version.

The Boolean circuit to be evaluated privately has  $g$  gates,  $u$  inputs and  $o$  outputs and has size  $N = u + g$ . The circuit is assumed to be built of only two-input NAND gates so that their functionality does not need to be hidden. There exist established highly optimized hardware synthesis tools that optimize for a small number of NAND gates when translating the function to a circuit. Moreover, it is assumed that “*the output wires of the circuit do not connect to any other gates*” [KM11] which is achieved by adding at most  $o$  gates to the circuit. [KM11] define the wiring among the gates as follows: Incoming wires are the inputs of the  $g$  gates. Outgoing wires are the output wires of the  $g$  gates and the  $u$  input wires of the circuit. Each incoming wire must be connected to exactly one outgoing wire, but an outgoing wire may be connected to more incoming wires, enabling gates with arbitrary fan-out. In contrast, UC-based PFE requires the fan-out to be at most two which requires additional copy-gates [Val76] that increase the circuit size.

Party  $P_2$  inputs private data  $x$  of length  $|x| = u$  and acts as the *circuit garbler* from Yao’s protocol.  $P_1$  inputs the private circuit  $\mathcal{C}_f$  of  $g$  gates and acts as the *circuit evaluator*. Since  $P_2$  must remain unaware of the circuit wiring,  $P_2$  cannot directly garble the gates. Instead,  $P_1$  creates a so-called encrypted garbled gate  $\text{encGG}_i$  for each gate  $i$  of the circuit and  $P_2$  decrypts these to learn

the keys required to create the garbled tables as in Yao's protocol (cf. Sect. 3.1 and [LP09]). By creating the encrypted garbled gates under HE,  $P_1$  obviously connects two *outgoing wires* to each gate of the circuit (the wire keys for the outgoing wires are provided by  $P_2$  beforehand). Thereby, the circuit topology remains hidden from  $P_2$ .

**Four Phases of PFE Protocols.** We split the protocol of [KM11] and UC-based PFE into four phases: 1) a *precomputation* phase which is run only once, 2) a  $setup_N$  phase dependent on the size  $N$  of the function, 3) a  $setup_f$  phase dependent on the function  $f$ , and 4) an  $online_x$  phase dependent on the input  $x$ .

$P_1$ (inputs private circuit $\mathcal{C}_f$ )	$P_2$ (inputs private data $x \in \{0, 1\}^u$ )
<b>1) precomputation phase</b>	
$\xleftarrow{\text{pk}}$	$(\text{pk}, \text{sk}) \xleftarrow{\$} \text{KGen}(1^\kappa)$ $r \xleftarrow{\$} \{0, 1\}^\kappa$
<b>2) <math>setup_N</math> phase</b>	
$\forall i \in \{1, \dots, g\} :$ $b_i, b'_i \xleftarrow{\$} \{0, 1\}^\kappa$ , compute $\text{Enc}_{\text{pk}}(b_i), \text{Enc}_{\text{pk}}(b'_i)$	$\forall i \in \{1, \dots, N\} :$ $(s_i^0) \xleftarrow{\$} \{0, 1\}^\kappa$ $\forall i \in \{1, \dots, N - o\} :$ compute $\text{Enc}_{\text{pk}}(s_i^0)$
$\xleftarrow{\text{Enc}_{\text{pk}}(s_1^0), \dots, \text{Enc}_{\text{pk}}(s_{N-o}^0)}$	
<b>3) <math>setup_f</math> phase</b>	
$\forall i \in \{1, \dots, g\} :$ gate $i$ has left input $j$ , right input $k$ $v_{iL} = \text{Enc}_{\text{pk}}(s_j^0 + b_i)$ $v_{iR} = \text{Enc}_{\text{pk}}(s_k^0 + b'_i)$ $\text{encGG}_i = (v_{iL}, v_{iR})$	$\forall i \in \{1, \dots, g\} :$ gate $i$ has output wire $u+i$ $L_i^0 = \text{Dec}(v_{iL}), L_i^1 = L_i^0 + r$ $R_i^0 = \text{Dec}(v_{iR}), R_i^1 = R_i^0 + r$ $GT_i = \text{encYao}(L_i^0, L_i^1, R_i^0, R_i^1, i, s_{u+i}^0, s_{u+i}^1)$
$\xrightarrow{\text{encGG}_1, \dots, \text{encGG}_g}$	
$\xleftarrow{GT_1, \dots, GT_g}$	
<b>4) <math>online_x</math> phase</b>	
$\forall i \in \{1, \dots, g\} :$ $L_i = s_j + b_i, R_i = s_k + b'_i$ $s_{u+i} = \text{decYao}(L_i, R_i, i, GT_i)$	$\xleftarrow{s_1^{x_1}, \dots, s_u^{x_u}}$ $\xrightarrow{s_{N-o+1}, \dots, s_N}$ compare $s_{N-o+1}, \dots, s_N$ to $(s_{N-o+1}^0, s_{N-o+1}^1), \dots, (s_N^0, s_N^1)$ to determine output $f(x)$

**Fig. 1.** The [KM11] protocol. The circuit  $\mathcal{C}_f$  has  $u$  input wires,  $o$  output wires,  $g$  gates, and size  $N = u + g$ . The symmetric security parameter is  $\kappa = 128$ .

In most applications, e.g., when a server provides a service with a pre-defined function (such as privacy-preserving IDS, cf. Sect. 1), the *precomputation* and both *setup* phases can be precomputed before the client provides its input, allowing for a very fast *online<sub>x</sub>* phase. In other applications, the function may not be known beforehand, in which case the *precomputation* and *setup<sub>N</sub>* phases can be precomputed, and the *setup<sub>f</sub>* and *online<sub>x</sub>* phases are run online.

**1) *precomputation phase.*** We first determine all operations that have to be done once, independently of the protocol run: For [KM11], this includes generating and sending the public key of the HE scheme, and for UC-based PFE, the construction of the UC itself. We do not include this phase in our performance evaluation in Sect. 5.

**2) *setup<sub>N</sub> phase.*** This phase precomputes all operations that depend only on the size  $N$  of the circuit. In [KM11],  $P_2$  creates two wire keys representing the bit values 0 and 1 for each of the  $N = g + u$  outgoing wires. The wire keys of all  $g + u$  outgoing wires except the  $o$  output wires of the circuit are essential to define the mapping representing the topology of the circuit. We denote the wire key corresponding to the bit value  $b \in \{0, 1\}$  on outgoing wire  $i \in \{1, \dots, N\}$  by  $s_i^b$ . The security of the protocol depends on the indistinguishability of the two keys.  $P_2$  chooses the wire key  $s_i^0$  at random and, similar to the *free-XOR* technique [KS08b], defines a global random shift  $r$  of the same size as the wire keys.  $P_2$  then sets  $s_i^1 = s_i^0 + r$  for  $i \in \{1, \dots, N\}$  and sends the homomorphically encrypted wire keys  $\text{Enc}(s_1^0), \dots, \text{Enc}(s_{N-o}^0)$  to  $P_1$ . As a preparation for the *setup<sub>f</sub>* phase,  $P_1$  already creates and encrypts two random blinding values,  $b_i$  and  $b'_i$ , for each gate  $G_i$ . This phase has complexity  $\mathcal{O}(N)$ .

In the UC-based PFE protocols, the UC is garbled and sent to the evaluator, which has complexity  $\Theta(N \log N)$ .

**3) *setup<sub>f</sub> phase.*** This depends on the specific function  $f$ . In [KM11], party  $P_1$  creates the encrypted garbled gates. In order to hide the wiring of the circuit from  $P_2$ , each wire key is blinded. If outgoing wires  $j$  and  $k$  are connected to the incoming wires of gate  $G_i$ ,  $P_1$  constructs the encrypted garbled gate  $\text{encGG}_i$  by making use of the additively homomorphic property of  $\text{Enc}$  as

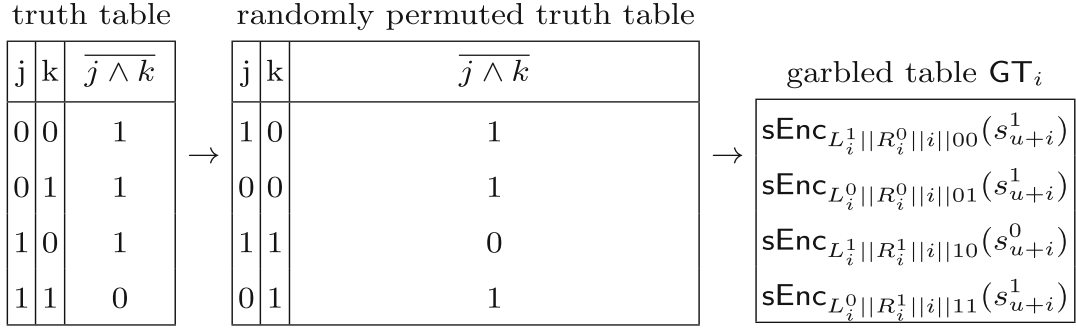
$$\text{encGG}_i = (\text{Enc}(s_j^0 + b_i), \text{Enc}(s_k^0 + b'_i)). \quad (1)$$

$P_1$  then sends  $\text{encGG}_1, \dots, \text{encGG}_g$  to  $P_2$ .  $P_2$  is now able to create the garbled tables and thereby acts as the *circuit garbler* from Yao's protocol. For each gate  $G_i$ ,  $P_2$  decrypts the corresponding encrypted garbled gate and retrieves the blinded wire keys for the left and the right incoming wire of the gate:

$$L_i^0 = \text{Dec}(\text{Enc}(s_j^0 + b_i)), \quad R_i^0 = \text{Dec}(\text{Enc}(s_k^0 + b'_i)). \quad (2)$$

$P_2$  is now able to obtain the blinded wire keys  $s_j^1 + b_i$  and  $s_k^1 + b'_i$  by defining  $L_i^1 = L_i^0 + r$  and  $R_i^1 = R_i^0 + r$ . Note that the blinded wire keys  $L_i^0, L_i^1$  and  $R_i^0, R_i^1$  are independent of the keys assigned to the outgoing wires of gates  $j$  and  $k$ . This hides the circuit topology from  $P_2$  while still enabling  $P_2$  to create the garbled





**Fig. 2.** `encYao`: creation of a garbled table [LP09]

tables. The garbled table  $\text{GT}_i$  is generated using function `encYao`, instantiated as shown in Fig. 2 [LP09]: The truth table of the NAND gate is randomly permuted and then for each combination of the left  $(L_i^0, L_i^1)$  and right  $(R_i^0, R_i^1)$  input key these keys are used to symmetrically encrypt the output key  $s_{u+i}^0$  or  $s_{u+i}^1$  using function `sEnc` which is instantiated using AES-128 (cf. Sect. 5.1 for details). We emphasize that the gates' output keys are pre-determined and the protocol of [KM11] applies additively homomorphic operations on input keys. Therefore, we cannot use GC optimizations like point-and-permute [BMR90], garbled row reduction [NPS99, Pin+09], or half-gates [ZRE15]. Instead, we have to use the classical GC from [LP09] with four entries per garbled table (GT), so each GT has size  $4 \cdot (|s_{u+i}| + \sigma)$  bits, where  $\sigma = 40$  is the statistical security parameter. Finally,  $P_2$  sends  $\text{GT}_1, \dots, \text{GT}_g$  to  $P_1$ . This phase has complexity  $\mathcal{O}(N)$ .

In the UC-based PFE protocols, the wire keys specifying the values of the UC's programming bits are sent which yields  $\Theta(N \log N)$  communication.

**4) *online<sub>x</sub>* phase.** In this final phase, the private data  $x$  is input by  $P_2$ . In [KM11], the wire keys  $s_1^{x_1}, \dots, s_u^{x_u}$  of the circuit input wires corresponding to  $P_2$ 's input bits  $x_1, \dots, x_u$  are sent to  $P_1$ .<sup>4</sup>  $P_1$  can now evaluate the garbled tables and determine the wire keys of the output wires as follows: To evaluate gate  $i$ ,  $P_1$  has to reconstruct the keys used to encrypt one entry of the garbled table. Starting with the first gate in topological order,  $P_1$  uses for gate  $G_i$  with left input  $j$  and right input  $k$  the keys  $s_j \in \{s_j^0, s_j^1\}$  and  $s_k \in \{s_k^0, s_k^1\}$  and the blinding values  $b_i, b'_i$  from the  $\text{setup}_N$  phase to calculate  $L_i = s_j + b_i$  and  $R_i = s_k + b'_i$ .  $P_1$  now decrypts the garbled table  $\text{GT}_i$  to learn the wire key  $s_{u+i} = \text{decYao}(L_i, R_i, i, \text{GT}_i)$  as in Yao's garbled circuit protocol and continues with the next gate in topological order. Once all gates have been evaluated,  $P_1$  has obtained the wire keys  $s_{N-o+1}, \dots, s_N$  of the output wires. These can be mapped to plaintext outputs as in Yao's protocol. However, as mentioned in Sect. 3.2, the function holder  $P_1$  should not learn the output of  $f$ , so the output is determined by party  $P_2$ . This phase has complexity  $\mathcal{O}(N)$ .

<sup>4</sup> The protocol can naturally be extended to the setting where also  $P_1$  has private input data  $y$ . Either  $y$  is encoded in the private function  $f$  [PSS09], or the keys corresponding to the bits of  $y$  are obviously sent to  $P_1$  using oblivious transfer [Ish+03, LP09, Ash+13] as describe in [KM11].



In the UC-based PFE protocols, the wire keys corresponding to the private input  $x$  are sent, the garbled UC is evaluated, which requires  $\Theta(N \log N)$  computation, and the output bits of the UC are decoded.

## 4.2 Optimizations of the [KM11] Protocol

In this section, we describe our optimizations to the protocol of [KM11].

**Precomputation of All Homomorphic Encryptions.** As described in Sect. 4.1, all homomorphic encryptions can be precomputed in the  $setup_N$  phase where only the size  $N$  is known but neither  $\mathcal{C}_f$  nor  $x$ . Since encryption is a relatively expensive operation, this drastically reduces the protocol runtime (see Sect. 5.2).

The wire keys are sampled randomly so depend neither on the inputs nor on the circuit  $\mathcal{C}_f$ , and are encrypted using the HE public key generated by  $P_2$ .

$P_2$  can sample and homomorphically encrypt the *encrypted wire keys*  $\text{Enc}(s_i^0)$ , where  $1 \leq i \leq N$ . Similarly,  $P_1$  can sample and encrypt the blinding values  $b_i, b'_i$ , where  $1 \leq i \leq g$ , using  $P_1$ 's public key. Here, it is necessary to exchange the public key of the HE scheme first. We argue that this is feasible in practice by  $P_2$  publishing the public key beforehand.

**Pipelining.** The creation and evaluation of the garbled circuit (GC) is done in topological order which makes this process eligible for pipelining. When transmitting the garbled gates directly after creation, they can be ungarbled by the evaluator while subsequent gates are still being garbled by the garbler. This GC pipelining was proposed and implemented in [Hen+10, Hua+11].

In addition to the GC pipelining, we also implemented pipelining of the creation and evaluation of the encrypted garbled gates. The process of retrieving the wire keys from the encrypted garbled gates can then seamlessly be combined with the pipelined creation and evaluation of the GC. Since decryption of the encrypted garbled gates is the most expensive operations in the  $setup_f$  phase, this significantly speeds up the protocol and reduces the time spent solely on network communication. In our experiments, we saw that pipelining improves the runtime in the  $setup_f$  phase by about 25%.

**Parallelization.** The [KM11] protocol is very suitable for parallelization. We provide a fully parallelized implementation of 1) the creation of the encrypted wire keys by  $P_2$  and the encrypted blinding values by  $P_1$  in the  $setup_N$  phase, 2) the creation of the encrypted garbled gates by  $P_1$  in the  $setup_f$  phase, 3) the decryption of the encrypted garbled gates and the creation of the garbled tables by  $P_2$  in the  $setup_f$  phase. Only the evaluation of the garbled tables by  $P_1$  depends on the wire keys obtained from previous garbled tables and therefore cannot be fully parallelized.

## 4.3 Instantiating [KM11] with EC ElGamal

Katz and Malka suggest to use ElGamal encryption to instantiate their protocol [KM11], and briefly mention the possibility of using elliptic curve

cryptography (ECC) in their protocol. In the following, we denote integers by lowercase letters and points on the elliptic curve by capital letters. The equivalent of choosing a random element of the residue field as the private key in standard ElGamal encryption is choosing a random integer  $a$  from the Galois field  $GF(p)$  as the private key in the elliptic curve version. The public key  $A$  is then computed as  $A = a * P$  where  $P$  is the base point of the elliptic curve.

In standard additively homomorphic lifted EC ElGamal, a message  $m \in GF(p)$  is mapped to a curve point  $M$  as  $M = m * P$ . The reverse mapping used during decryption then requires solving the discrete logarithm of  $M$  which requires that  $m$  is from a small domain whereas we need to operate on  $\kappa = 128$  bit keys. Instead, we observe that the only requirement for the choice of the wire keys and the blinding values in the [KM11] protocol is indistinguishability, so we can simply define curve points  $M$  as our plaintext values for wire keys and blinding values. Then, we perform plaintext additions using the ECC arithmetic on the elliptic curve when  $P_1$  needs to apply the blinding value to a plaintext wire key in order to determine the values  $L_i$  and  $R_i$ . These points are then mapped to keys for AES using a KDF (cf. Sect. 5.1).

Analogous to standard ElGamal, we define encryption of a message  $M$  with a public key  $A = a * P$  as follows:

$$\text{Enc}(M) = (K, C) = (k * P, k * A + M). \quad (3)$$

Decryption of the ciphertext  $(K, C)$  can now be done as follows:

$$\text{Dec}(K, C) = C - a * K = k * A + M - a * k * P = k * a * P + M - a * k * P = M. \quad (4)$$

EC ElGamal is additively homomorphic in the underlying elliptic curve group. We define the homomorphic addition of two ciphertexts as

$$\text{Enc}(M_1) \oplus \text{Enc}(M_2) = (K_1, C_1) \oplus (K_2, C_2) = (K_1 + K_2, C_1 + C_2). \quad (5)$$

This satisfies the additively homomorphic property over the EC group:

$$\begin{aligned} \text{Dec}(\text{Enc}(M_1) \oplus \text{Enc}(M_2)) &= \text{Dec}((k_1 * P, k_1 * A + M_1) \oplus (k_2 * P, k_2 * A + M_2)) \\ &= \text{Dec}((k_1 * P + k_2 * P, k_1 * A + M_1 + k_2 * A + M_2)) \\ &= \text{Dec}((k_1 + k_2) * P, (k_1 + k_2) * A + M_1 + M_2) \\ &= (k_1 + k_2) * A + M_1 + M_2 - a * (k_1 + k_2) * P = M_1 + M_2. \end{aligned} \quad (6)$$

Semantic security naturally follows from that of ElGamal based in the DDH assumption in the EC group.

#### 4.4 Instantiating [KM11] with BFV Homomorphic Encryption

Since the linear-complexity protocol of [KM11] was proposed in 2011, significant progress has been made in the area of Ring-LWE (RLWE) based homomorphic encryption. Thus, we revise the protocol of [KM11] with an HE instantiation

based on these efficient Ring-LWE HE schemes. We specifically use the BFV scheme (cf. Sect. 3.3) as implemented in Microsoft’s SEAL library [Sea19]. We take the plaintext modulus as  $t = 2$ , which results in the smallest possible polynomial modulus degree and thus ciphertext size in our scenario. The coefficient modulus  $q$  is chosen as a product of primes  $q_1 = 12289$  and  $q_2 = 1099510054913$ .  $q_1$  is the smallest prime that is large enough to allow homomorphic blinding of the key values and satisfies  $q_1 \equiv 1 \pmod{2n}$ , where  $n$  is *polynomial modulus degree* (cf. [Sea19] for details). For function privacy, which is necessary to prevent  $P_2$  from learning the permutation of the keys employed by  $P_1$ , we flood the ciphertext with noise (cf. [Lai17, §9.4]) that is 40-bits larger than the noise of the output ciphertext, ensuring a statistical security of 40-bits against  $P_2$ . Thus, we require an additional 40-bits (in the form of  $q_2$ ) in the coefficient modulus to contain the extra noise. Consequently, we choose  $p = 2048$  as the polynomial modulus degree, which is the smallest  $n$  that maintains computational security of 128-bits for a  $q$  of 54-bits (cf. [Lai17], Table 3).

**Encoding of the Wire Keys.** When choosing a plaintext modulus of  $t = 2$ , each bit of the plaintext value is encoded as one coefficient of the polynomial. Assume we have a wire key  $v$  with a binary representation of  $v = v_{127}||v_{126}||\dots||v_0$ , we define our plaintext polynomial as  $v_{127}x^{127} + \dots + v_1x + v_0$ . Since homomorphic addition is done coefficient-wise in the BFV scheme and we use  $t = 2$ , addition becomes equivalent to a *homomorphic XOR operation*.

Due to the requirement that each wire key has to be utilized separately when creating the encrypted garbled gates, Chinese Remainder Theorem (CRT) batching, as provided by SEAL, becomes inefficient for our use case. Using batching, one can pack  $n$  integers modulo  $t$  into one plaintext polynomial and apply SIMD (Single Instruction, Multiple Data) operations on those values. However, this would require a much larger value for  $t$ . A multiplication operation (by a one-hot encoded vector), that is needed to extract one wire key from the ciphertext containing  $n$  wire keys, is less efficient than encrypting and decrypting a smaller ciphertext on its own. We therefore decided against CRT batching.

**Efficient Packing of the Ciphertexts.** The encoding of the wire keys uses exactly 128 coefficients of the BFV ciphertext. Since the degree of the polynomial modulus (`poly_modulus_degree`) is set to 2048, we only use  $\frac{1}{16}$  of the coefficients of each ciphertext. Even though we decided not to use CRT batching, utilizing the unused coefficients for packing additional 15 wire keys in a ciphertext seems desirable in order to reduce the communication of the protocol by a factor of 16.

Unfortunately, without access to the secret key, it is not possible for  $P_1$  to homomorphically extract a subset of coefficients of the underlying plaintext, and thus a wire key. Therefore, multiple wire keys can only be packed in a response to  $P_2$  holding the secret key.

Traditionally, each of the encrypted garbled gates consists of two ciphertexts, holding the blinded wire keys for the two incoming wires of that gate. First, we describe a way to combine the encrypted wire keys,  $\text{Enc}(s_j)$  and  $\text{Enc}(s_k)$ , into one ciphertext  $\text{Enc}(s_j||s_k)$ . Since in the plaintexts the wire keys of length 128-bits are followed by  $15 \times 128$  coefficients set to zero, we can use these coefficients to encode

further wire keys. We achieve this by applying a “homomorphic (right) bit shift” of 128-bits (respectively coefficients) to one of the wire keys (by multiplying a ciphertext by the plaintext constant  $2^{128}$ ) and adding both wire keys afterwards.

These wire keys still have to be blinded to form the encrypted garbled gate  $\text{encGG}_i$ , which can now be achieved by only one homomorphic addition. Therefore, we concatenate the blinding values  $b_i$  and  $b'_i$  and homomorphically add them to  $\text{Enc}(s_j||s_k)$  to receive the encrypted garbled gate  $\text{encGG}_i = \text{Enc}(s_j||s_k) + \text{Enc}(b_i||b'_i) = \text{Enc}((s_j + b_i)||s_k + b'_i)$ . Since  $P_2$  is in charge of telling the wire keys apart, “unpacking” is simply done by decrypting the ciphertext and assigning 128-bits to both wire keys.

Analogously, we can pack additional encrypted garbled gates into the same ciphertext and thereby use all 2048 coefficients to pack 8 encrypted garbled gates. This can be done efficiently using Horner’s method as described in [KSS13]. Blinding of the wire keys can now be applied by concatenating 16 blinding values and add them to the ciphertext in a single homomorphic addition.

Compared to not using this packing technique, we require the same number of homomorphic additions (15 additions to pack the 16 wire keys + 1 addition for the combined blinding value instead of one addition of a blinding value per wire key) and 15 multiplications by  $2^{256}$ , but we also eliminated 15 decryptions since  $P_2$  only receives one ciphertext instead of 16. Since for our instantiation of the BFV protocol decryption is more expensive than homomorphic scalar multiplication, this also improves computation.

**Wire Key Generation Using Seed Expansion.** The wire keys are encrypted by the private key owner  $P_2$  and can be homomorphically encrypted using the *secret* key to have smaller noise and smaller ciphertext size. When encrypting with the secret key, half of the ciphertext coefficients are chosen uniformly at random from  $R_q$ . Using a pseudo-random function, one can sample these coefficients by expanding a seed sent to  $P_1$  instead. This nearly halves the ciphertext size of the encrypted wire keys and significantly improves communication which is the major bottleneck of the scheme.<sup>5</sup>

## 5 Evaluation

In this section, we describe our implementation of the different instantiations of the [KM11] protocol and point out bottlenecks and advantages. We experimentally compare our implementations with the best existing UC-based PFE implementation of [Alh+20]. We also give estimates on the efficiency of the recent UC improvements of [Liu+20] that results in 33% smaller UCs and hence would improve UC-based PFE of [Alh+20] by around 33% in both runtime and communication (cf. dashed lines in Fig. 4 and 3). The results of our performance tests show that HE-based linear-complexity PFE supersedes UC-based PFE in runtime starting from a few thousand gates already and in communication for all

<sup>5</sup> Since January 2020 (version 3.4.0) the SEAL library [Sea19] supports seed expansion and encryption with the secret key. Our implementation uses this optimization.

circuit sizes. Hence, linear-complexity PFE is a viable alternative for improving the performance of private function evaluation.

## 5.1 Implementation

We implemented our optimized and fully parallelized version of the [KM11] protocol described in Sect. 4 using the ABY SFE framework [DSZ15]. Our implementation is available as open-source at <https://encrypto.de/code/linearPFE>. This is the first implementation of a linear-complexity PFE protocol. We provide a fair comparison with today’s most efficient UC-based PFE implementation of [Alh+20] with complexity  $\Theta(N \log N)$  which is based on the same STPC framework ABY.

We instantiate  $\text{sEnc}$  as  $\text{sEnc}_{k'}(m) = (AES_{k'}(0) || AES_{k'}(1) || \dots || AES_{k'}(\lceil (|m| + \sigma)/128 \rceil - 1)) \oplus (m || 0^\sigma)$ , where  $AES$  is AES-128 and  $\sigma = 40$  is the statistical security parameter. The arbitrary-length key  $k$  is mapped to a 128-bit key  $k' = \text{KDF}(k)$  where the KDF is instantiated with  $PBKDF2$ .

We instantiate the DJN cryptosystem with modulus size of 3072 bits.

In our EC ElGamal-based implementation we use the eBATS B-251 binary elliptic curve. RELIC encodes each point on the elliptic curve in 33 bytes.

SEAL serializes ciphertexts as 64-bit values using a compression function. For our specific choice of parameters, this compression did not achieve ideal results. For all ciphertexts except the encrypted wire keys where a seed is used to reduce their size, we implemented our own serialization where we eliminate unnecessary zeroes and thereby reduce the ciphertext size compared to the SEAL encoding.

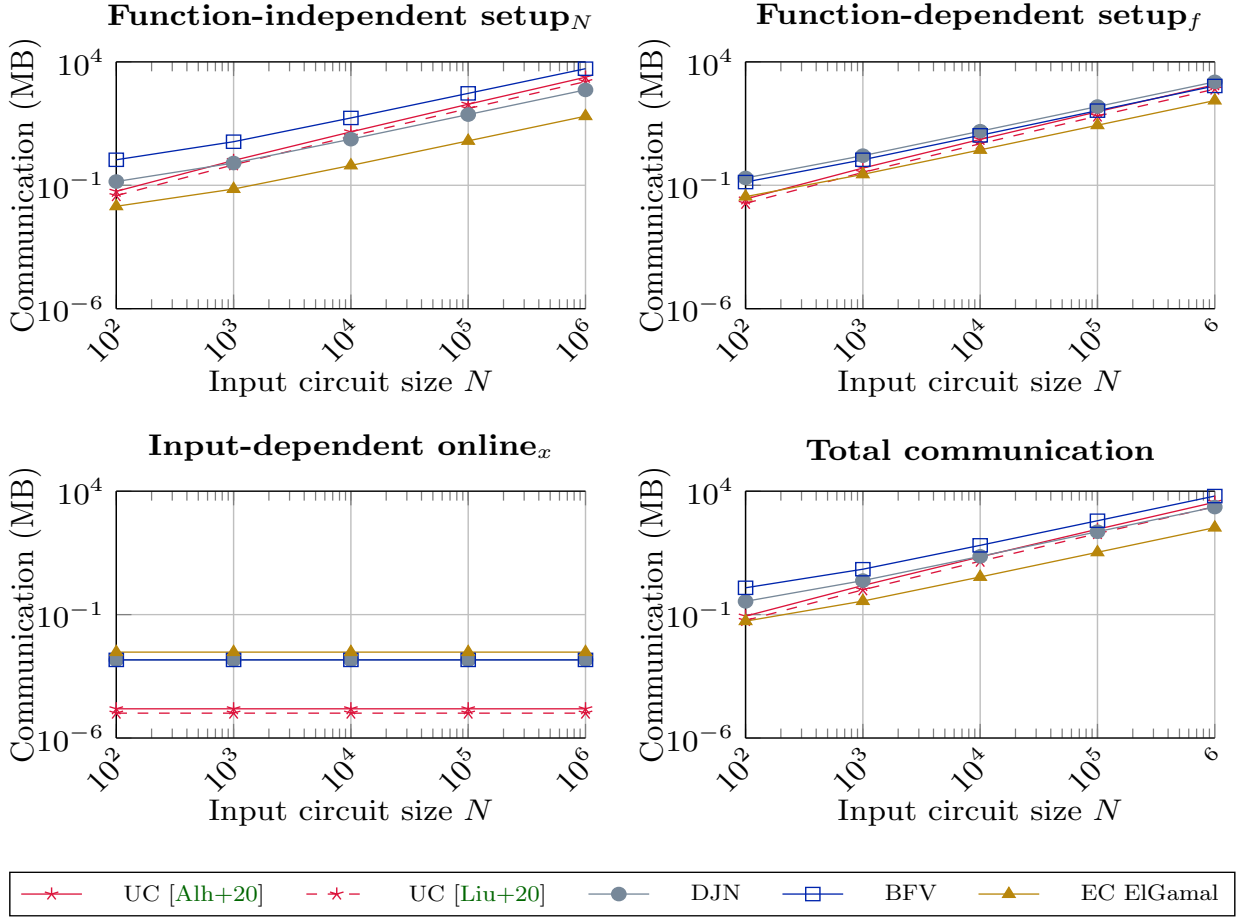
## 5.2 Experimental Evaluation

We use two identical machines with a physical connection of 10 Gbit/s bandwidth and a round-trip time of 1 ms. We refer to this as the LAN setting and also simulated a WAN setting with 100 Mbit/s bandwidth and a round-trip time of 100 ms. Each machine is equipped with an Intel Core i9-7960X CPU (32 Cores, 2.8 GHz) and 128 GB RAM. All measurements are averaged over 10 executions. Because in all PFE protocols the costs for the input  $x$  is substantially lower than for the gates, we fix the number of input bits to  $u = |x| = 64$ . The exact performance measures used to plot the figures are given in the full version [Hol+20].

**Communication.** In Fig. 3, we depict the communication of the PFE protocols. The EC ElGamal instantiation clearly outperforms all other implementations, including UC-based PFE [Alh+20] and thereby offers the best PFE scheme in terms of communication known so far. Its communication is lower than UC-based PFE of [Alh+20] by a factor of  $\sim 11\times$  for circuit size  $N = 10^6$ .

We observe that the communication complexity of DJN-based PFE is on par with UC-based approaches. Due to its large ciphertext size, BFV-based encryption has the worst communication of our instantiations but it is only a factor of about  $1.8\times$  higher for  $N = 10^6$  than that of UC-based PFE [Alh+20]. Its communication is significantly reduced by the seed expansion technique to





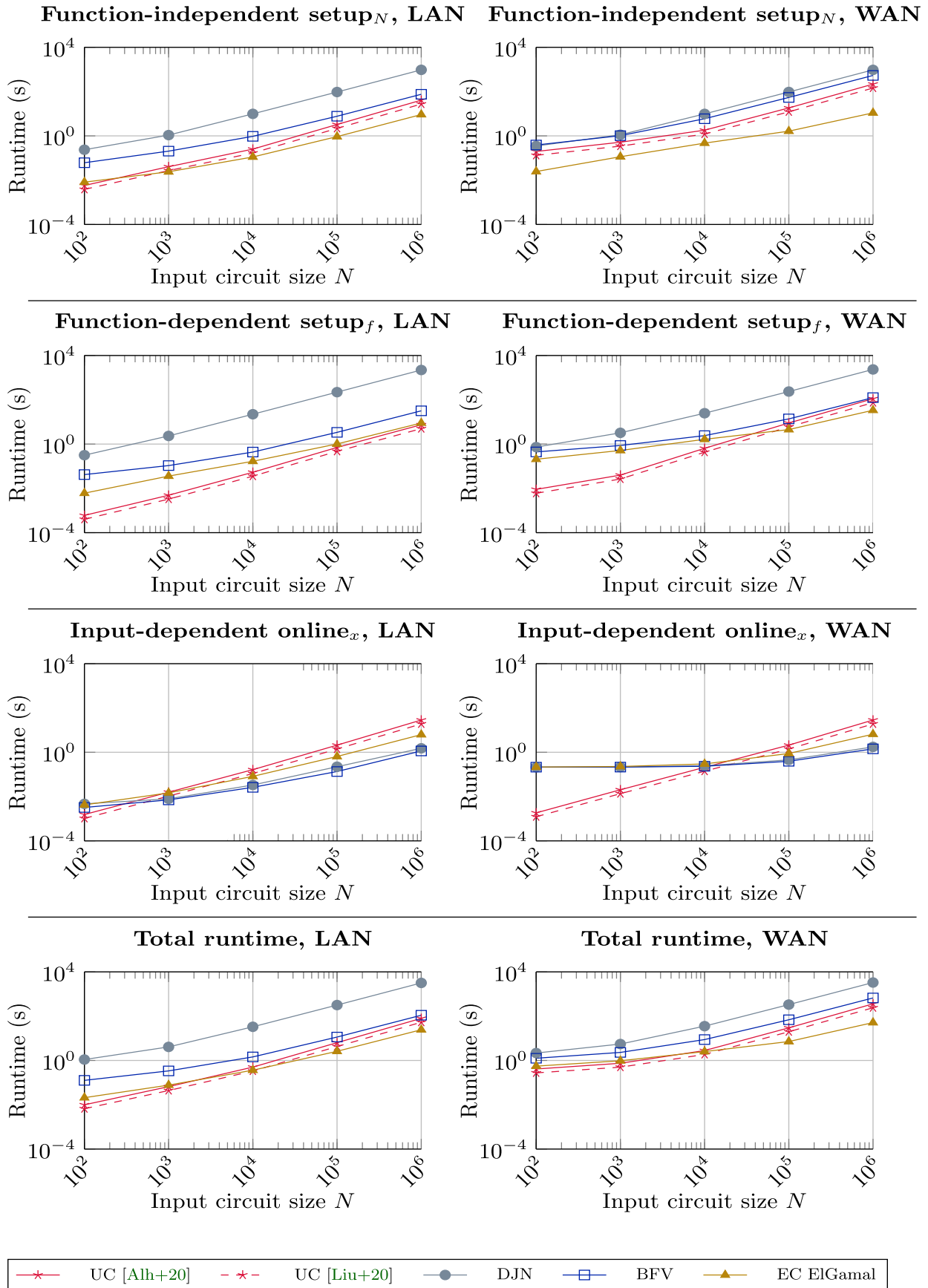
**Fig. 3.** Communication of PFE protocols (in MB).

reduce the size of the encrypted wire keys in the BFV scheme (cf. Sect. 3.3). In the  $online_x$  phase, the communication of all protocols only depends on the size of the input  $x$  and is nearly negligible (only a few KB).

**Runtime.** In Fig. 4, we depict the runtime of our implementation compared to the most recent UC-based PFE implementation of [Alh+20].

*ECC-based PFE* is our fastest implementation: Compared to the state-of-the-art UC-based PFE implementation of [Alh+20], the total runtime for  $N = 10^6$  gates is faster by a factor  $\sim 3.3\times$  in LAN and  $\sim 7.0\times$  in WAN.

*BFV-based PFE* offers promising total runtimes even though it is *less* efficient than ECC-based PFE of [Alh+20] by a factor of  $\sim 1.4\times$  in LAN and  $\sim 1.8\times$  in WAN for  $N = 10^6$ . The larger factor in the WAN setting results from its larger communication overhead compared to ECC-based PFE. These findings underline that though computational complexity is still relevant, communication complexity becomes the bottleneck for these PFE protocols. Therefore, the computational advances of BFV cannot compensate its larger ciphertext sizes any more. Still, our implementation instantiated with the BFV scheme beats [Alh+20] for circuits of about  $N \geq 250000$  gates when function- and input-independent pre-computations from the  $setup_N$  phase are excluded cf. full version [Hol+20]).

**Fig. 4.** Runtime of PFE protocols (in seconds).



*DJN-based PFE* has impractical computational overhead, i.e., about 53 minutes of runtime for  $N = 10^6$  gates in LAN (compared to 24s of the ECC-based instantiation), even with the optimizations described in Sect. 3.3. Its runtime in WAN is similar to WAN as it is dominated by computation.

**Per-phase Comparison.** In the  $setup_N$  phase, computation and communication are independent of the function  $f$  and input  $x$  and only depend on the (maximum) size of  $f$ . This yields significant large precomputation capabilities of HE-based PFE, especially for our BFV-based instantiation.

In the  $setup_f$  phase, the logarithmic overhead of UC-based PFE of [Alh+20] has a large performance impact. In contrast, HE-based protocols scale linearly and outperform UC-based PFE for  $N \geq 10^6$  in LAN and  $N \geq 250000$  in WAN.

In the  $online_x$  phase, HE-based PFE outperforms UC-based PFE of [Alh+20] for about  $N \geq 1000$  gates in LAN and  $N \geq 10000$  gates in WAN. Here, the computation is dominated by GC evaluation. The logarithmic overhead of the UC size compared to the actual circuit leads to a noticeable performance drawback. Since our ECC-based implementation uses points on the elliptic curve as wire keys (encoded as 264 bit values), the GC is larger by a factor of about two compared to the BFV- and DJN-based instantiations where wire keys have size 128 bits. This impacts GC evaluation runtime and BFV-based PFE becomes the fastest instantiation in the  $online_x$  phase.

When excluding precomputation of the  $setup_N$  phase from the total runtime, BFV-based PFE outperforms UC-based PFE of [Alh+20] for about  $N \geq 250000$  in LAN and WAN, and ECC-based PFE outperforms [Alh+20] for about  $N \geq 10000$  in LAN and about  $N \geq 25000$  in WAN (cf. full version [Hol+20]).

**Summary.** In this paper, we optimize and implement the linear-complexity PFE protocol of [KM11]. Our elliptic curve ElGamal-based implementation outperforms the state-of-the-art UC-based PFE implementation of [Alh+20] not only in communication, but also in total runtime: For private circuits of size  $N = 10^6$ , our implementation is  $\sim 3.3\times$  faster in a LAN and  $\sim 7.0\times$  faster in a WAN setting and scales with  $\mathcal{O}(N)$  instead of  $\Theta(N \log N)$ .

**Acknowledgement.** This project received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230, and by BMBF and HMWK within ATHENE.

## References

- [AG09] Aranha, D.F., Gouvêa, C.: RELIC cryptographic toolkit (2009). <https://github.com/relic-toolkit>
- [Alh+20] Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. J. Cryptol. **33**(3), 1216–1271 (2020). <https://doi.org/10.1007/s00145-020-09346-z>

- [Ash+13] Asharov, G., Lindell, Y., Schneider, T., Zohner M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS 2013, pp. 535–548. ACM (2013)
- [Bel+13] Bellare, M., Hoang, V.T., Keelveedhi, S., Rogaway, P.: Efficient garbling from a fixed-key blockcipher. In: S&P 2013, pp. 478–492. IEEE (2013)
- [BGV12] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Innovations in Theoretical Computer Science (ITCS 2012), pp. 309–325. ACM (2012)
- [Biç+18] Biçer, O., Bingöl, M.A., Kiraz, M.S., Levi, A.: Highly efficient and reusable private function evaluation with linear complexity. Cryptology ePrint Archive, Report 2018/515 (2018). <https://ia.cr/2018/515>
- [Bin+18] Bingöl, M.A., Biçer, O., Kiraz, M.S., Levi, A.: An efficient 2-party private function evaluation protocol based on half gates. Comput. J. **62**(4), 598–613 (2018)
- [BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: STOC 1990, pp. 503–513. ACM (1990)
- [Bra12] Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32009-5\\_50](https://doi.org/10.1007/978-3-642-32009-5_50)
- [DJN10] Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of Paillier’s public-key system with applications to electronic voting. Int. J. Inf. Secur. **9**(6), 371–385 (2010). <https://doi.org/10.1007/s10207-010-0119-9>
- [DSZ15] Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: NDSS 2015. The Internet Society (2015)
- [Elg85] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. Trans. Inf. Theory **31**(4), 469–472 (1985)
- [FAL06] Frikken, K.B., Atallah, M.J., Li, J.: Attribute-based access control with hidden policies and hidden credentials. IEEE Trans. Comput. **55**(10), 1259–1270 (2006)
- [FAZ05] Frikken, K.B., Atallah, M.J., Zhang, C.: Privacy-preserving credit checking. In: ACM Conference on Electronic Commerce (EC 2005), pp. 147–154. ACM (2005)
- [Fel+19] Felsen, S., Kiss, Á., Schneider, T., Weinert, C.: Secure and private function evaluation with Intel SGX. In: CCSW 2019, pp. 165–181. ACM (2019)
- [FV12] Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). <https://ia.cr.org/2012/144>
- [GKS17] Günther, D., Kiss, Á., Schneider, T.: More efficient universal circuit constructions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 443–470. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-70697-9\\_16](https://doi.org/10.1007/978-3-319-70697-9_16)
- [GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game. In: STOC 1987, pp. 218–229. ACM (1987)
- [Gün+19] Günther, D., Kiss, Á., Scheidel, L., Schneider, T.: Framework for semi-private function evaluation with application to secure insurance rate calculation. CCS 2019 Posters/Demos (2019)
- [Hen+10] Henecka, W., Kögl, S., Sadeghi, A.-R., Schneider, T., Wehrenberg, I.: TASTY: tool for automating secure two-party computations. In: CCS 2010, pp. 451–462. ACM (2010)

- [HMS12] Hu, Y., Martin, W.J., Sunar, B.: Enhanced flexibility for homomorphic encryption schemes via CRT. In: ACNS 2012 (Industrial Track) (2012)
- [Hol+20] Holz, M., Kiss, Á., Rathee, D., Schneider, T.: Linear-complexity private function evaluation is practical (full version). Cryptology ePrint Archive, Report 2020/853 (2020). <https://ia.cr/2020/853>
- [Hua+11] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster secure two-party computation using garbled circuits. In: USENIX Security 2011. USENIX (2011)
- [IR89] Impagliazzo, R., Rudich, S.: Limits on the provable consequences of one-way permutations. In: STOC 1989, pp. 44–61. ACM (1989)
- [Ish+03] Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003). [https://doi.org/10.1007/978-3-540-45146-4\\_9](https://doi.org/10.1007/978-3-540-45146-4_9)
- [KM11] Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 556–571. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-25385-0\\_30](https://doi.org/10.1007/978-3-642-25385-0_30)
- [Kob87] Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177), 203–209 (1987)
- [KS08a] Kolesnikov, V., Schneider, T.: A practical universal circuit construction and secure evaluation of private functions. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 83–97. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-85230-8\\_7](https://doi.org/10.1007/978-3-540-85230-8_7)
- [KS08b] Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-70583-3\\_40](https://doi.org/10.1007/978-3-540-70583-3_40)
- [KS16] Kiss, Á., Schneider, T.: Valiant’s universal circuit is practical. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9665, pp. 699–728. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49890-3\\_27](https://doi.org/10.1007/978-3-662-49890-3_27)
- [KSS13] Kolesnikov, V., Sadeghi, A.-R., Schneider, T.: A systematic approach to practically efficient general two-party secure function evaluation protocols and their modular design. J. Comput. Secur. **21**(2), 283–315 (2013)
- [Lai17] Laine, K.: Simple encrypted arithmetic library 2.3.1. Microsoft Research (2017). <https://www.microsoft.com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1.pdf>
- [Liu+20] Liu, H., Yu, Y., Zhao, S., Zhang, J., Liu, W.: Pushing the limits of Valiant’s universal circuits: simpler, tighter and more compact. Cryptology ePrint Archive, Report 2020/161 (2020). <https://ia.cr/2020/161>
- [LMS16] Lipmaa, H., Mohassel, P., Sadeghian, S.S.: Valiant’s universal circuit: improvements, implementation, and applications. Cryptology ePrint Archive, Report 2016/17 (2016). <https://ia.cr/2016/017>
- [LP09] Lindell, Y., Pinkas, B.: A proof of security of Yao’s protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009). <https://doi.org/10.1007/s00145-008-9036-8>
- [LPR10] Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-13190-5\\_1](https://doi.org/10.1007/978-3-642-13190-5_1)

- [Mil86] Miller, V.S.: Use of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986). [https://doi.org/10.1007/3-540-39799-X\\_31](https://doi.org/10.1007/3-540-39799-X_31)
- [MS13] Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-38348-9\\_33](https://doi.org/10.1007/978-3-642-38348-9_33)
- [MSS14] Mohassel, P., Sadeghian, S., Smart, N.P.: Actively secure private function evaluation. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 486–505. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-45608-8\\_26](https://doi.org/10.1007/978-3-662-45608-8_26)
- [Nik+14] Niksefat, S., Sadeghian, B., Mohassel, P., Sadeghian, S.: ZIDS: a privacy-preserving intrusion detection system using secure two-party computation protocols. *Comput. J.* **57**(4), 494–509 (2014)
- [NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: ACM Conference on Electronic Commerce (EC 1999), pp. 129–139. ACM (1999)
- [Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48910-X\\_16](https://doi.org/10.1007/3-540-48910-X_16)
- [Pin+09] Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-10366-7\\_15](https://doi.org/10.1007/978-3-642-10366-7_15)
- [PSS09] Paus, A., Sadeghi, A.-R., Schneider, T.: Practical secure evaluation of semi-private functions. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 89–106. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-01957-9\\_6](https://doi.org/10.1007/978-3-642-01957-9_6)
- [Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC 2005, pp. 84–93. ACM (2005)
- [Sea19] Microsoft SEAL (release 3.3) (2019). <https://github.com/Microsoft/SEAL>
- [Val76] Valiant, L.G.: Universal circuits (preliminary report). In: STOC 1976, pp. 196–203. ACM (1976)
- [Yao82] Yao, A.C.: Protocols for secure computations (extended abstract). In: FOCS 1982, pp. 160–164. IEEE (1982)
- [Yao86] Yao, A.C.-C.: How to generate and exchange secrets. In: FOCS 1986, pp. 162–167. IEEE (1986)
- [Zha+19] Zhao, S., Yu, Yu., Zhang, J., Liu, H.: Valiant’s universal circuits revisited: an overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, vol. 11921, pp. 401–425. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-34578-5\\_15](https://doi.org/10.1007/978-3-030-34578-5_15)
- [ZRE15] Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46803-6\\_8](https://doi.org/10.1007/978-3-662-46803-6_8)

## E SoK: Modular and Efficient Private Decision Tree Evaluation (PoPETs'19)

---

- [KNL<sup>+</sup>19] Á. KISS, M. NADERPOUR, J. LIU, N. ASOKAN, T. SCHNEIDER. “**SoK: Modular and Efficient Private Decision Tree Evaluation**”. In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2019.2 (2019). De Gruyter Open. Online version: <https://ia.cr/2018/1099>. Code: <https://crypto.de/code/PDTE>, pp. 187–208. CORE Rank B. Appendix E.

<https://doi.org/10.2478/popets-2019-0026>



Ágnes Kiss\*, Masoud Naderpour, Jian Liu, N. Asokan, and Thomas Schneider

# SoK: Modular and Efficient Private Decision Tree Evaluation

**Abstract:** Decision trees and random forests are widely used classifiers in machine learning. Service providers often host classification models in a cloud service and provide an interface for clients to use the model remotely. While the model is sensitive information of the server, the input query and prediction results are sensitive information of the client. This motivates the need for private decision tree evaluation, where the service provider does not learn the client's input and the client does not learn the model except for its size and the result.

In this work, we identify the three phases of private decision tree evaluation protocols: feature selection, comparison, and path evaluation. We systematize constant-round protocols for each of these phases to identify the best available instantiations using the two main paradigms for secure computation: garbling techniques and homomorphic encryption. There is a natural tradeoff between runtime and communication considering these two paradigms: garbling techniques use fast symmetric-key operations but require a large amount of communication, while homomorphic encryption is computationally heavy but requires little communication. Our contributions are as follows: Firstly, we systematically review and analyse state-of-the-art protocols for the three phases of private decision tree evaluation. Our methodology allows us to identify novel combinations of these protocols that provide better tradeoffs than existing protocols. Thereafter, we empirically evaluate all combinations of these protocols by providing communication and runtime measures, and provide recommendations based on the identified concrete tradeoffs.

**Keywords:** Privacy-preserving protocols, secure computation, garbling techniques, homomorphic encryption, decision tree evaluation, machine learning

DOI 10.2478/popets-2019-0026

Received 2018-08-31; revised 2018-12-15; accepted 2018-12-16.

**\*Corresponding Author: Ágnes Kiss:** TU Darmstadt, E-mail: kiss@encrypto.cs.tu-darmstadt.de

**Masoud Naderpour:** University of Helsinki, E-mail: masoud.naderpour@helsinki.fi

**Jian Liu:** University of California, Berkeley, E-mail: liu.jian@berkeley.edu (This work was done when the author was in Aalto University)

## 1 Introduction

Machine learning is pervasively being applied in various real-world scenarios. All major IT companies, including Amazon, Apple, Google, Facebook and Microsoft, are working on and use machine learning technologies. First, a *training phase* takes place where a model is trained on a large dataset so that later it can make predictions on an input, which happens in the *evaluation phase*. One of the most commonly used predictive models are decision trees, whose extension to random forests – sets of smaller decision trees trained on different random subsets of features – is considered to be among the most accurate classification models [DCBA14].

In many scenarios, predictive models such as decision trees and neural networks are trained or queried on sensitive data and should be handled in a private manner [WGC18]. Solutions for *private training* of decision trees exist that have been heavily optimized [LP00, LP02, VC05, VCKP08, BS09, FW12].

In this work, we focus on the *private evaluation* of decision trees with security against passive adversaries. In this scenario, the server holds a decision tree and offers its evaluation as a service to its client with sensitive input to the decision tree. Moreover, the model may leak information about the training data used for its generation, which is a valuable asset of the server. The server's goal is to offer the service without compromising the client's input, or the client learning anything about the decision tree (beyond its size and the result).

Note that recent works [TZJ<sup>+</sup>16, PMG<sup>+</sup>17, SSSS17, JSD<sup>+</sup>18] have shown that it is possible for an attacker who has only blackbox oracle access to prediction APIs of machine learning models to succeed in learning model parameters by repeated adaptive queries to the API. Defending against such attacks is an active area of research and several proposals have already appeared, e.g., [TZJ<sup>+</sup>16, KMAM18, JSD<sup>+</sup>18]. We deem dealing with blackbox attacks out of scope for this paper.

**N. Asokan:** Aalto University, E-mail: asokan@acm.org

**Thomas Schneider:** TU Darmstadt, E-mail: schneider@encrypto.cs.tu-darmstadt.de

## 1.1 Applications

Cloud-hosted evaluation of machine learning models is a widely used service that companies such as Amazon, Google or IBM provide to their customers. While doing so [Bar15], Amazon Web Services (AWS) recognizes that their clients care about the privacy of their data [Ser18]. These services can be enhanced by using private evaluation of machine learning models to allow for providing a service while violating the privacy of neither the resulting model nor the customers’ input features. Most machine learning APIs such as BigML [Big18], MLJAR [MLJ17], Wise.io [Wis18], or Azure Machine Learning Studio [Mic18] support decision tree or random forest classifiers. Decision tree evaluation has a tremendous amount of applications, and is being used by companies such as Facebook [IK17] and Microsoft [RG16].

Private decision tree evaluation can provide a solution for private medical diagnosis [TASF09], such as electrocardiogram classification [BFK<sup>+</sup>09, BFL<sup>+</sup>11], where the medical history and genomic data of the patient are sensitive information. Decision trees and random forests can be applied for malware [AM18], and text classification [RMD18]. Moreover, decision trees and random forests have been successfully used for detecting insider threat [MAAG15] and multimedia protocol tunneling [BSR18], as well as spam campaigners [GKG<sup>+</sup>18], and predicting the interruptability intensity of mobile users [YGL17], data hotspots in cellular networks [NIZ<sup>+</sup>16], and even private interactions in underground forums [OTGM18]. Private evaluation of decision trees and random forests can enhance the privacy that these techniques provide, while protecting the sensitive business information of the service provider. Recently, secure evaluation of branching programs, an extension of decision trees, has also been utilized in a protocol for private set intersection [CO18].

## 1.2 Outline and Our Contributions

There exist two main paradigms for secure computation: *homomorphic encryption*, which often has little communication but high computation complexity, and *garbling techniques*, which require more communication, but utilize efficient symmetric-key primitives. In this work, we systematically analyze the state-of-the-art constant-round protocols for private decision tree evaluation that make use of these paradigms. We explore the tradeoffs and combinations of the identified sub-protocols, compare the performance of all combinations and identify

Sub-protocol	Path <sub>H</sub>	Path <sub>G</sub>
Sel <sub>H</sub> + Comp <sub>G</sub>	HGH §4.1	HGG [BFK <sup>+</sup> 09]
Sel <sub>G</sub> + Comp <sub>G</sub>	GGH §4.1	GGG [BFK <sup>+</sup> 09]
Sel <sub>H</sub> + Comp <sub>H</sub>	HHH [TMZC17]	HHG §4.1
Sel <sub>G</sub> + Comp <sub>H</sub>	×	×

**Table 1.** Combinations of sub-protocols for private decision tree evaluation  $ABC = \text{Sel}_A + \text{Comp}_B + \text{Path}_C$ . Sel<sub>H</sub>/Sel<sub>G</sub> denotes oblivious feature selection, Comp<sub>H</sub>/Comp<sub>G</sub> oblivious comparison, and Path<sub>H</sub>/Path<sub>G</sub> oblivious path evaluation with homomorphic encryption (H) and garbling techniques (G).

the most efficient constant-round private decision tree evaluation protocols to date.

In more details, our contributions are as follows:

**Systematization and analysis of existing techniques (§3).** Private decision tree evaluation consists of three sub-protocols: The first is *private selection* of features that are to be compared with in each of the decision nodes, the second is *private comparison*, and the third is *private evaluation of the path* corresponding to the given input in the decision tree. We analyze the state-of-the-art techniques for these sub-protocols that use additively homomorphic encryption (H) as well as those using garbling techniques (G). We denote the different approaches for oblivious selection, comparison, and path evaluation by Sel<sub>H</sub>/Sel<sub>G</sub> (§3.1), Comp<sub>H</sub>/Comp<sub>G</sub> (§3.2), and Path<sub>H</sub>/Path<sub>G</sub> (§3.3), respectively, and describe their efficient instantiations.

**Protocols for private decision tree evaluation (§4).** We show that different sub-protocols can be combined efficiently, resulting in the six protocols shown in Tab. 1. We adapt all resulting protocols to an offline-online setting, where all operations independent of the input features are precomputed in an offline phase, which results in a more efficient online phase (§4.1). Thereafter, we present protocol extensions for some of our protocols (§4.2).

**Empirical evaluation (§5).** We evaluate the performance of the resulting private decision tree evaluation protocols and study the tradeoff between runtime and communication on example real-world datasets. We show the competitive performance of general-purpose secure two-party computation protocols for sub-protocols of private decision tree evaluation that has been claimed to be inefficient in [WFNL16, TMZC17].

**Improvements and recommendations (§6).** We show that our identified hybrid protocols that combine homomorphic encryption with garbling techniques achieve an order of magnitude runtime improvement



$S$	Server (holds decision tree)
$C$	Client (holds input features)
$T$	Decision tree
$n$	Dimension of feature vector
$t$	Bitlength of a feature
$m$	Number of decision nodes
$m'$	Number of depth-padded decision nodes
$\overline{m}$	Number of possibly depth-padded decision nodes, i.e., $m$ or $m'$ , depending on the underlying protocol
$d$	Depth of decision tree
$\mathbf{x} = \{x_1, \dots, x_n\}$	Client's feature vector
$\mathbf{y} = \{y_1, \dots, y_{\overline{m}}\}$	Server's thresholds for (padded) decision nodes
$\kappa$	Symmetric security parameter (= 128)
$s$	Statistical security parameter (= 40)
$\tau_{\text{sym}}$	Size of symmetric ciphertext (= 128)
$\tau_{\text{ElGamal}}$	Size of ciphertext in lifted ElGamal (= 514)
$\tau_{\text{Paillier}}$	Size of ciphertext in Paillier (= 4096)
$\tau_{\text{DGK}}$	Size of ciphertext in DGK encryption (= 2048)

**Table 2.** Notations used throughout the paper.

over state-of-the-art protocols. We provide recommendations for different settings based on our findings.

**Related work (§7) and open questions.** Finally, we discuss our results in relation to related work. The protocol based solely on garbling techniques can easily be extended to security against malicious clients (cf. §4.1). Tai et al. [TMZC17] propose the same (more costly) extension using zero-knowledge proofs for the protocol based solely on homomorphic encryption. As interesting future work, one could study and extend the security of hybrid protocols against malicious clients.

## 2 Preliminaries

In this section, we detail necessary preliminaries to our work. The notations we use are summarized in Tab. 2.

### 2.1 Decision Trees

A decision tree is a binary tree  $T$  with  $m$  internal nodes called *decision nodes*, as shown in the example in Fig. 1a. Every leaf node is called a *classification node* and is associated with a classification value  $\mathbf{v} = \{v_1, \dots, v_{m+1}\}$ . A decision tree has a threshold vector  $\mathbf{y} = \{y_1, \dots, y_m\}$  and receives an  $n$ -dimensional feature vector  $\mathbf{x} = \{x_1, \dots, x_n\}$  as input. At the  $j^{\text{th}}$  decision node ( $j \in \{1, \dots, m\}$ ), a comparison between an  $x_i$  and  $y_j$  takes place ( $i \in \{1, \dots, n\}$ ), where  $\sigma : m \rightarrow n$  denotes the mapping for input selection. Decision tree

evaluation  $T(\mathbf{x})$  means evaluating the comparisons at each decision node of a path, the result of which defines the branch taken next. When a leaf node is reached on the path, the corresponding classification is outputted. The *depth*  $d$  of the decision tree is the length of the longest path between the root and any leaf.

Additionally, we define *depth-padding*, where dummy decision nodes are introduced at every leaf until depth  $d$  is reached in order for each path to have the same depth. A dummy node has one outgoing edge, i.e., independently from the comparison result it leads to the same node as shown in Fig. 1b. The number of these dummy nodes, which depends on the structure of the tree, is at most  $\sum_{i=1}^{m-1} i$ , i.e., the number of decision nodes in the depth-padded tree is  $m \leq m' \leq \frac{1}{2}m(m+1)$ . From here on, we use  $\overline{m}$  whenever both  $m$  or  $m'$  can be used in a protocol.

**Real-world datasets.** In this paper, we use the real-world datasets summarized in Tab. 3 for which we trained decision trees using scikit-learn [sld17]. These datasets are taken mostly from the UCI machine learning repository [Lic18]: the classical iris dataset for pattern recognition of the iris flower, the wine and (breast) cancer datasets for classification based on the chemical analysis of wines and the characteristics of a diagnostic image, resp., as well as the boston dataset that includes house prices, the digits dataset that contains images of hand-written digits, which has been used by NIST [GBC<sup>+</sup>97], and the diabetes dataset that includes automatic and paper records of diabetes patient glucose measurements for classification. The linnerud dataset has been used in [Ten98] for regression and includes physiological measurements. The decision trees are of varying sizes and depths which allows us to study the behaviour of our identified protocol combinations. In Tab. 3, we show the values of  $n$  (number of input features),  $d$  (depth),  $m$  (number of decision nodes), and  $m'$  (number of depth-padded decision nodes).

### 2.2 Cryptographic Techniques

Private decision tree evaluation protocols can be constructed based on both paradigms for secure computation: homomorphic encryption and garbling techniques. We detail the techniques used in this work.

**Oblivious transfer.** 1-out-of-2 oblivious transfer (OT) allows a sender to obliviously send one of two messages  $m_0, m_1$  to a receiver according to the receiver's selection bit  $b$ , without the receiver learning anything about  $m_{1-b}$  or the sender learning  $b$ . OT generally

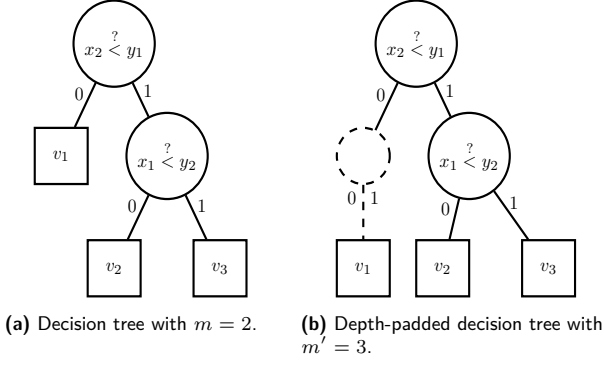


Fig. 1. Decision trees with depth  $d = 2$ .

Dataset	$n$	$d$	$m$	$m'$	Source
iris	3	5	8	19	UCI [Lic18]
wine	7	5	11	26	UCI [Lic18]
linnerud	3	6	19	47	[Ten98]
(breast) cancer	12	7	21	66	UCI [Lic18]
digits	47	15	168	1,161	UCI [Lic18]
diabetes	10	28	393	6,432	UCI [Lic18]
boston	13	30	425	6,768	UCI [Lic18]

Table 3. Example decision trees of varying sizes with  $n$  input features,  $m$  decision nodes,  $m'$  padded decision nodes, and  $d$  depth for real-world datasets, trained using `scikit-learn` [sld17] which contains a library with these datasets.

requires expensive public-key operations [IR89]. However, a small (symmetric security parameter  $\kappa$ ) number of base OTs can be extended to any polynomial number of OTs efficiently using only symmetric-key operations [IKNP03]. Improvements to such OT extension protocols have been presented with security against passive [ALSZ13] and active adversaries [ALSZ15, KOS15]. OTs can be efficiently precomputed [Bea95], which is suited for an offline-online scenario, where the parties can precompute parts of the protocol before providing their inputs to the computation: During precomputation in the offline phase, it is possible to run random OTs on random inputs, which are then used to mask the actual messages and choice bits. To retrieve the output of the OT in the online phase, the parties blind their messages using inexpensive XOR operations, and require  $|m_0| + |m_1| + 1$  bits of communication, whereas all cryptographic operations are shifted to the offline phase, which requires  $\tau_{\text{sym}}$  bits of communication per random OT [ALSZ13].

**Yao’s garbled circuit protocol.** Secure two-party computation allows two parties to compute an arbitrary functionality on their private inputs without learning anything about the other party’s input (beyond the output). Yao’s garbled circuit (GC) proto-

col [Yao82, Yao86] is one of the most widely used solutions for secure two-party computation. Yao’s protocol works as follows: one of the parties acts as the garbler, who garbles the circuit, i.e., assigns two random keys corresponding to 0 and 1 to each of its wires and encrypts the gates using these keys. The garbler sends this garbled circuit  $GC$  to the other party, the evaluator, who evaluates it after having obviously received the keys corresponding to its input bits from the garbler through OTs. The protocol therefore relies on OTs linear in the input length of the evaluator and symmetric-key operations linear in the size of the Boolean circuit describing the function. Various optimizations exist including point-and-permute [BMR90], free-XOR [KS08a], fixed-key AES garbling [BHKR13], and half-gates [ZRE15], while at the implementation level Single Instructions Multiple Data (SIMD) operations can be used [BJL12]. All these state-of-the-art optimizations are implemented in the ABY framework [DSZ15], which we use in our implementation. With all these optimizations, each AND gate of the circuit needs  $2\tau_{\text{sym}}$  bits of communication.

Additionally, we use the point-and-permute technique in our protocols [BMR90]. In the classic garbling scheme, each wire  $i$  has two labels ( $k_i^0$  for 0 and  $k_i^1$  for 1) associated with it. In order to hide the truth values of these wires with minimal overhead, a *color bit* can be appended to each label (opposite color bits for 0 and 1). The relation of the color bits and the truth values are known only to the garbler, while the evaluator holds one label from each wire, the color bit of which it can use to proceed with evaluation. In the end of the protocol, the evaluator possesses the wire labels for the output value, which have the color bits appended to them.

We use the following notations for the garbling scheme:  $(GC, \text{out}^0, \text{out}^1) \leftarrow \text{Garble}(C, \text{in}^0, \text{in}^1)$  denotes the garbling method that returns the garbled circuit  $GC$ , including the output wire keys  $(\text{out}^0, \text{out}^1)$  for both the 0 and 1 values, respectively. It gets as input circuit  $C$  and input wire key pairs  $(\text{in}^0, \text{in}^1)$  for the evaluator’s input wires.  $(GC, \text{color}) \leftarrow \text{Garble}'(C, \text{in}^0, \text{in}^1)$  denotes the same garbling method but returns the color bits along with  $GC$ .  $\text{out} \leftarrow \text{Eval}(GC, \text{in})$  denotes GC evaluation where the evaluator inputs his input wire keys  $\text{in}$  (received via OTs from the garbler), and receives the output wire keys  $\text{out}$  corresponding to the output bits. The actual output remains oblivious to the evaluator, since he does not learn the values the wire keys correspond to.  $\text{out}' \leftarrow \text{Eval}'(GC, \text{in})$  denotes a similar method, where the evaluator receives blinded output bits  $\text{out}'$ , i.e., the color bits of the outputs.

**Homomorphic encryption.** Additively homomorphic encryption allows anyone holding the public key  $pk$  to calculate  $[x_1 + x_2] = [x_1] \boxplus [x_2]$ , given two ciphertexts  $[x_1] := \text{Enc}_{pk}(x_1)$  and  $[x_2] := \text{Enc}_{pk}(x_2)$ . Multiplying  $[c \cdot x] = c \boxtimes [x]$  with constant  $c$  is supported since this is equal to adding  $[x]$  with itself  $c$  times, which can be efficiently computed using the double-and-add method.

We use semantically secure additively homomorphic encryption schemes. More specifically, we use Paillier encryption with packing [Pai99, DJ01], DGK encryption [DGK08], and Lifted ElGamal encryption [ElG85]. With a plaintext space of  $\mathbb{Z}_N$ , the ciphertext space of Paillier and DGK encryption are  $\mathbb{Z}_{N^2}^*$  and  $\mathbb{Z}_N^*$ , respectively, where  $N$  is an RSA modulus. Paillier encryption allows for packing of plaintexts and ciphertexts when the encrypted values are smaller than the plaintext space. We use ciphertext packing since the ciphertext  $[x]$  in our case is much larger than the plaintext  $x$ . Thus, we can pack  $n$  ciphertexts  $[x_1], \dots, [x_n]$  into a single one:  $[X] = [x_n || x_{n-1} || \dots || x_1]$ , and reduce the communication from  $n \cdot \tau_{\text{Paillier}}$  to  $n \cdot \tau_{\text{Paillier}} / |x_i|$ .

Lifted ElGamal encryption is similar to the classical ElGamal scheme, with the exception that the plaintext message is in the exponent, i.e.,  $m$  is replaced by  $g^m$ , where  $g$  is the group generator. This allows for additive homomorphism but makes decryption more difficult due to the discrete logarithm that needs to be solved. Lifted ElGamal can be instantiated using elliptic curve (EC) cryptography, where the ciphertext is represented by two EC points. This scheme is thus very efficient but can only be used when the encrypted plaintext values are in a known small subset of the plaintext space.

### 3 Protocol Building Blocks

Private decision tree evaluation is run between two parties: a server  $\mathcal{S}$  holding a classification tree  $T$  (with threshold vector  $\mathbf{y}$ , input selection  $\sigma$  and classification values  $\mathbf{v}$ ) and a client  $\mathcal{C}$  providing its input feature vector  $\mathbf{x}$ . After the protocol evaluation,  $\mathcal{C}$  obtains the classification result, without learning any information about the tree beyond its size and the output, and  $\mathcal{S}$  learns nothing about  $\mathcal{C}$ 's input features. The corresponding ideal functionality  $\mathcal{F}_{\text{PDTE}}$  is depicted in Fig. 2.

In this section, we recapitulate the state-of-the-art sub-protocols for private decision tree evaluation with security against semi-honest adversaries. We describe each protocol and refer to the original publications for specific details on each construction. In §3.1, we detail

Ideal functionality  $\mathcal{F}_{\text{PDTE}}$ .

1. **Private input of  $\mathcal{S}$ :** decision tree  $T$ .
2. **Private input of  $\mathcal{C}$ :** inputs  $\mathbf{x} = \{x_1, \dots, x_n\}$ .
3. **Compute:** evaluate  $v = T(\mathbf{x})$ .
4. **Output of  $\mathcal{C}$ :** classification result  $v$ .

**Fig. 2.**  $\mathcal{F}_{\text{PDTE}}$  – Ideal functionality for private decision tree evaluation (PDTE). A decision tree  $T$  is defined by its topology, input selection  $\sigma : \overline{m} \rightarrow n$ , thresholds  $\mathbf{y} = \{y_1, \dots, y_{\overline{m}}\}$  and classification values  $\mathbf{v} = \{v_1, \dots, v_{\overline{m}+1}\}$ .

Ideal functionality  $\mathcal{F}_{\text{Sel}}$ , given a secret sharing scheme  $S_1$ .

1. **Private input of  $\mathcal{S}$ :** selection  $\sigma : \overline{m} \rightarrow n$ .
2. **Private input of  $\mathcal{C}$ :** inputs  $\mathbf{x} = \{x_1, \dots, x_n\}$ .
3. **Compute:**  $\mathbf{z} = \{x_{\sigma(1)}, \dots, x_{\sigma(\overline{m})}\}$ , i.e., select  $\overline{m}$  values from the  $n$  inputs with selection  $\sigma$ .
4. **Output of  $\mathcal{S}, \mathcal{C}$ :** secret  $S_1$  shares of  $\mathbf{z}$ .

**Fig. 3.**  $\mathcal{F}_{\text{Sel}}$  – Ideal functionality for obliviously selecting  $\overline{m}$  elements from  $n \leq \overline{m}$  elements.

two protocols for privately selecting features from  $n$  input features that are assigned to a decision node to be compared at (*selection phase*). In §3.2, we recapitulate two sub-protocols for private comparison (*comparison phase*): one based on garbled circuits and another one based on additively homomorphic encryption. Thereafter in §3.3, we provide protocols for obliviously evaluating the path to the classification leaf based on the results of the previous phase (*path evaluation phase*).

In all sub-protocols presented in this section, the number of decision nodes in decision tree  $T$  is either  $m$  or the number of decision nodes  $m'$  after depth-padding (cf. §2.1). This is according to the approach chosen for path evaluation (§3.3). We use  $\overline{m}$  decision nodes in our description, where possibly  $\overline{m} = m$  or  $\overline{m} = m'$ .

For completeness, we give in §A the communication of all sub-protocols in Tab. 5 and their asymptotic computation complexities in Tab. 6.

Ideal functionality  $\mathcal{F}_{\text{Comp}}$ , given two secret sharing schemes  $S_1, S_2$ .

1. **Private inputs of  $\mathcal{S}, \mathcal{C}$ :** secret  $S_1$  shares of the selection  $x_{\sigma(1)} \dots, x_{\sigma(\overline{m})}$ .
2. **Private input of  $\mathcal{S}$ :**  $\mathbf{y} = \{y_1, \dots, y_{\overline{m}}\}$ .
3. **Compute:** for  $i \in \{1, \dots, \overline{m}\}$ :  $c_i = 1$  if  $x_{\sigma(i)} < y_i$ , else  $c_i = 0$ .
4. **Output of  $\mathcal{S}, \mathcal{C}$ :** secret  $S_2$  shares of  $c_1, \dots, c_{\overline{m}}$ .

**Fig. 4.**  $\mathcal{F}_{\text{Comp}}$  – Ideal functionality for obliviously comparing  $\overline{m}$  pairs of elements.

Ideal functionality  $\mathcal{F}_{\text{Path}}$ , given secret sharing scheme  $S_2$ .

1. **Private inputs of  $\mathcal{S}, \mathcal{C}$ :** secret  $S_2$  shares of the comparison results  $\{c_1, \dots, c_{\overline{m}}\}$ .
2. **Private input of  $\mathcal{S}$ :** decision tree  $T$ .
3. **Compute:** evaluation of the decision tree path using the comparison results resulting in leaf  $v$ .
4. **Output of  $\mathcal{S}$ :** classification result  $v$ .

**Fig. 5.**  $\mathcal{F}_{\text{Path}}$  – Ideal functionality for oblivious path evaluation.

### 3.1 Selection Phase

The selection phase obliviously selects from the  $n$  features of the client  $\mathcal{C}$  a value for each of the  $\overline{m}$  decision nodes (where they are to be compared to thresholds) according to the selection  $\sigma : \overline{m} \rightarrow n$  of the server  $\mathcal{S}$ . The ideal functionality  $\mathcal{F}_{\text{Sel}}$  that describes this selection is shown in Fig. 3. We identify two ways to instantiate it, one based on additively homomorphic encryption ( $\text{Sel}_H$ ), and another one based on the secure evaluation of a Boolean selection network using garbled circuits ( $\text{Sel}_G$ ). In both protocols, the outputs of the parties are secret shares of the result: none of the parties learns any information about the output, but they are able to continue secure computation with it.

**Selection using additively homomorphic encryption ( $\text{Sel}_H$ , Fig. 6).** Additively homomorphic encryption has been used to perform oblivious selection in [BPSW07, BFK<sup>+</sup>09]. In the protocol, the client  $\mathcal{C}$  encrypts its inputs, and sends the ciphertexts to the server  $\mathcal{S}$ .  $\mathcal{S}$  then selects the values according to selection  $\sigma$  and homomorphically blinds these with statis-

tical blinding using fresh randomness (pads are longer than the plaintext value by  $s$  bits, where  $s$  is the statistical security parameter).  $\mathcal{S}$  then sends these to  $\mathcal{C}$ , who decrypts them, retrieving a blinded selection of its inputs. In this sub-protocol, we use either Paillier encryption with ciphertext packing for the  $\overline{m}$  ciphertexts to reduce the communication to  $(n + \overline{m}/(t + s)) \cdot \tau_{\text{Paillier}}$  bits (cf. §2.2), or DKG encryption with smaller ciphertexts and  $(n + \overline{m}) \cdot \tau_{\text{DGK}}$  bits of communication.

**Selection using a garbled selection network ( $\text{Sel}_G$ , Fig. 7).** An alternative for obliviously selecting the features is evaluating a selection network [KS08b, BFK<sup>+</sup>09], which is based on Waksman’s permutation network [Wak68]. Firstly,  $\mathcal{S}$  sends the GC corresponding to the selection network to  $\mathcal{C}$ . They then perform  $nt$  OTs for each bit of the client’s input. In most cases,  $n \leq \overline{m}$ , since features are used more than once. The size of a selection network that maps  $n \leq \overline{m}$   $t$ -bit elements to  $\overline{m}$  elements is  $t \cdot S_{\overline{m} \geq n}^n = t \cdot (0.5(n + \overline{m}) \log(n) + \overline{m} \log(\overline{m}) - n + 1)$  [KS16, Appendix C]. The communication of this protocol is  $(n + 2S_{\overline{m} \geq n}^n)t \cdot \tau_{\text{sym}}$  bits offline and  $nt(2\tau_{\text{sym}} + 1)$  bits online.

### 3.2 Comparison Phase

This section details two oblivious comparison protocols, the ideal functionality  $\mathcal{F}_{\text{Comp}}$  of which is given in Fig. 4. The outputs of  $\mathcal{S}, \mathcal{C}$  are secret shares of the result. The first uses homomorphic encryption (instantiating both  $\text{Sel}_H$  and  $\text{Comp}_H$ ), the second uses Yao’s GC ( $\text{Comp}_G$ ).

**Comparison (including selection) using additively homomorphic encryption ( $\text{Sel}_H + \text{Comp}_H$ , Fig. 8).** This comparison protocol, often referred to as the DGK comparison protocol, was proposed in [DGK07, DGK08, DGK09] and used for private decision tree evaluation in [WFNL16, TMZC17]. Since the comparison protocol is performed using homomorphically encrypted bits of the client’s input and known thresholds, the selection phase  $\text{Sel}_H$  can be realized within this protocol without additional overhead. Thus, it already includes the oblivious selection protocol of §3.1 for free (cf. Tab. 1). It is observed in [DGK07] that  $x < y$  is true if and only if  $\exists i \in \{1, \dots, t\}$  such that  $f_i(x, y) = x_i - y_i + 1 + 3 \sum_{j < i} (x_j \oplus y_j) = 0$ .  $\mathcal{C}$  encrypts each bit of its input with an additively homomorphic encryption scheme (cf. §2.2), and sends the ciphertexts to  $\mathcal{S}$ , who chooses the appropriate feature to compare with at each decision node and generates a random bit  $a_k$  for each comparison ( $k \in \{1, \dots, \overline{m}\}$ ).

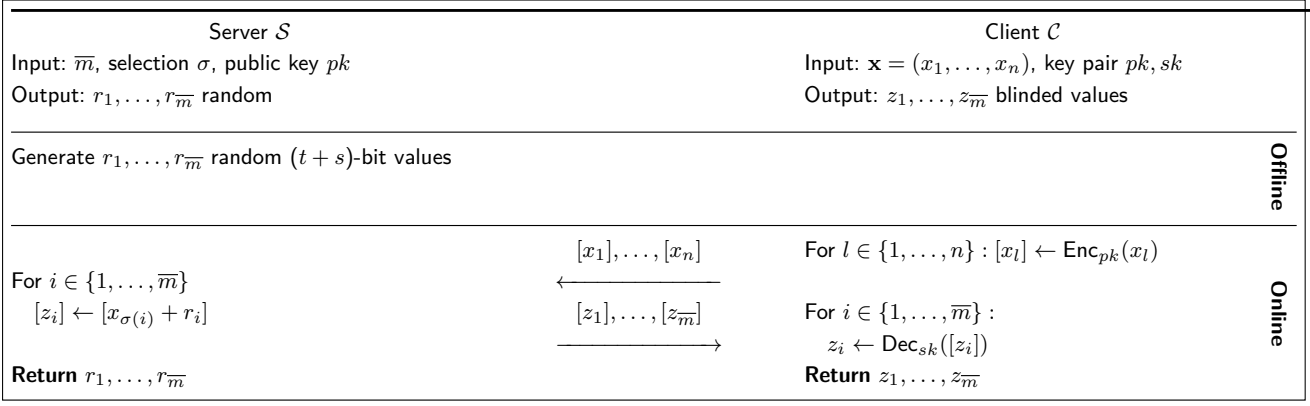


Fig. 6.  $\text{Sel}_H$  – Oblivious selection using additively homomorphic encryption.

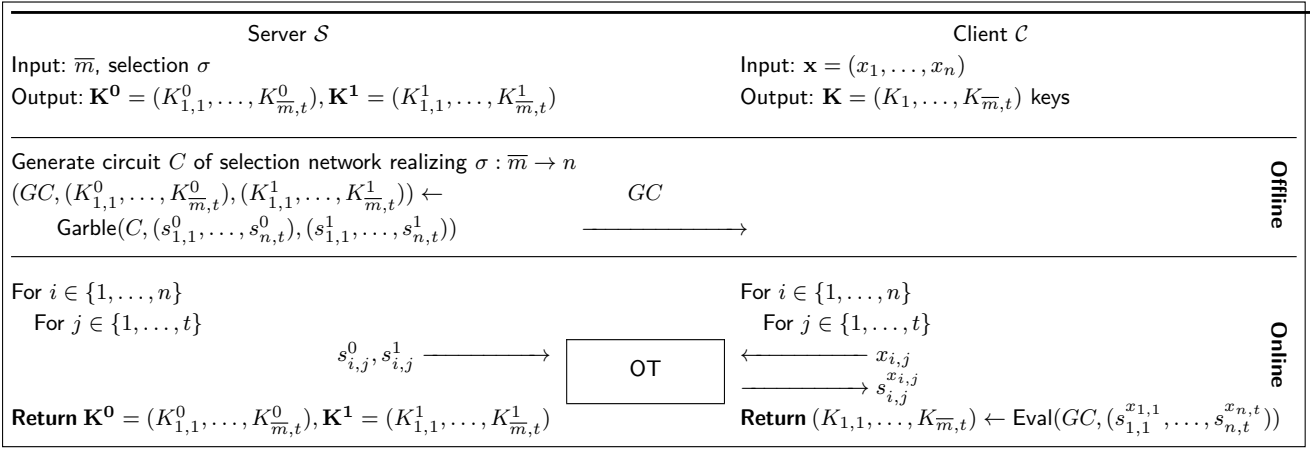


Fig. 7.  $\text{Sel}_G$  – Oblivious selection using a garbled selection network.

Then, the above comparison is performed that outputs a blinded ciphertext  $[r_i \cdot f_i(x, y)]$  for each bit of each feature. If any of these ciphertexts decrypts to zero, then  $x_i < y_i$ . The comparison bit is not revealed to the client in clear, but is secret shared between the parties ( $\mathcal{S}$  holds random bit  $a_k$ ,  $\mathcal{C}$  holds the result blinded by  $a_k$ , i.e.,  $b_k = a_k \oplus [x_{\sigma,k} < y_k]$ ). The efficient lifted ElGamal encryption scheme can be used here (cf. §2.2), since the client needs only to check if the result is  $g^0$  or not. The communication in this variant is  $(n + \overline{m})t \cdot \tau_{\text{ElGamal}}$  bits.

Joye and Salehi [JS18] present an optimization of the DGK comparison protocol that improves the communication roughly by factor two when comparing two values with each other. However, as opposed to the DGK comparison protocol presented in Fig. 8, it does not allow for including the selection step when more values are compared. This incurs an overhead when the client encrypts its elements, i.e., it now needs to encrypt  $\overline{m}$  instead of  $n$  elements. However, Joye and Salehi also propose a private decision tree evaluation protocol that uses their protocol such that only  $d$  comparisons be-

tween two values are performed as in plain decision tree evaluation. We compare with this protocol in §4.2.

**Comparison using garbled circuits (Comp<sub>G</sub>, Fig. 9).** All oblivious comparisons can also be performed with generic secure two-party computation protocols. This method has been used for comparison in [BPSW07, BFK<sup>+</sup>09]. Yao’s garbled circuit is particularly efficient for private comparison, and requires  $t$  AND gates per  $t$ -bit comparisons [KSS09]. Therefore,  $\overline{m}$  comparisons require  $2\overline{m}t \cdot \tau_{\text{sym}}$  bits communication. Variant (a) of this comparison protocol outputs the Boolean shares of the output of the comparisons, i.e., the color bits in Yao’s GC protocol (cf. §2.2). Variant (b) outputs the keys that correspond to the output wires: the server holds key pairs for both 0 and 1, while the client holds the keys for the output wires (without knowing the values they correspond to).

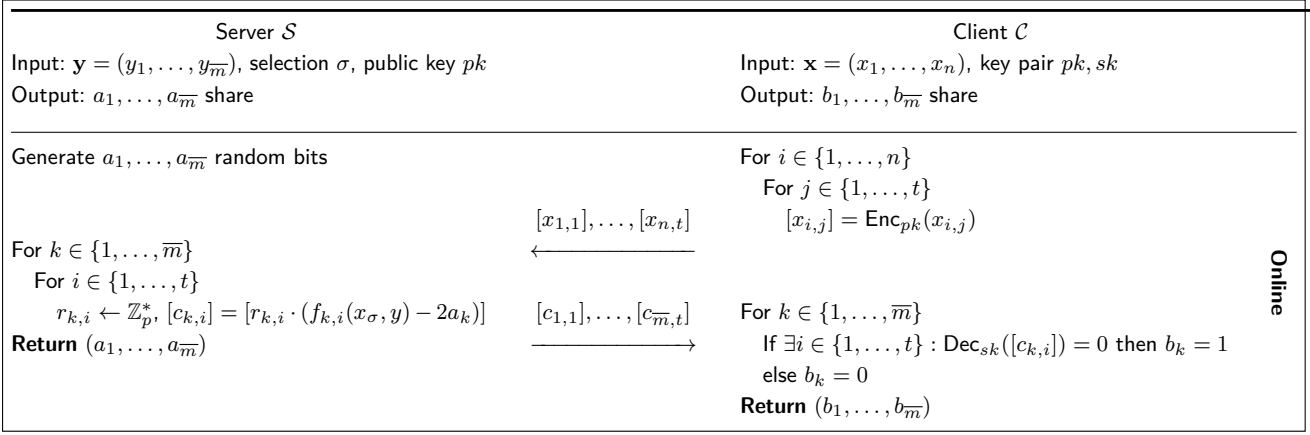


Fig. 8.  $\text{Sel}_H + \text{Comp}_H$  – Oblivious selection and comparison using additively homomorphic encryption from [WFNL16, TMZC17].

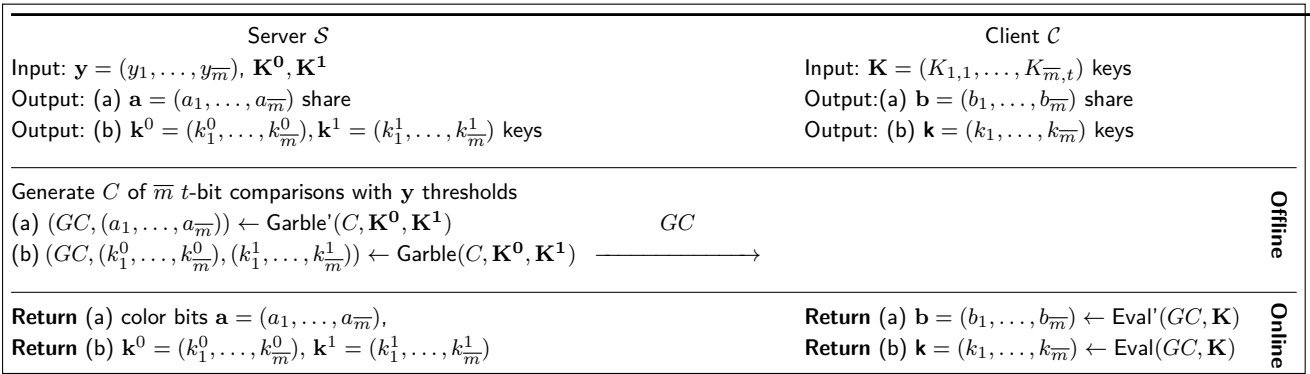


Fig. 9.  $\text{Comp}_G$  – Oblivious comparison using garbled circuits.

### 3.3 Path Evaluation Phase

This section describes two path evaluation protocols, the first using homomorphic encryption ( $\text{Path}_H$ ) and the second using a garbled decision tree ( $\text{Path}_G$ ). The ideal functionality  $\mathcal{F}_{\text{Path}}$  is given in Fig. 5.

**Path evaluation using additively homomorphic encryption ( $\text{Path}_H$ , Fig. 10).** The path evaluation protocol of [TMZC17] is highly optimized and depends only on the number of decision nodes  $m$ . The client  $\mathcal{C}$  encrypts his share of the comparison bit for each decision node, and sends it to the server. The server  $\mathcal{S}$  homomorphically computes the encryption of the actual comparison bit (by removing the blinding), which is the *edge cost* of the edge from the decision node that corresponds to 0 (left). The other edge, corresponding to 1 (right), gets the opposite of this bit as its edge cost. For each leaf node its *path cost* is computed by summing up the edge costs along its path from the root. This requires  $\mathcal{O}(m)$  operations since intermediate values can be reused. These and the classification labels on each leaf node are then blinded by  $\mathcal{S}$  using fresh randomness, and sent to the client  $\mathcal{C}$ . If the path cost decrypts to zero,  $\mathcal{C}$

can decrypt the classification label. This path evaluation protocol has  $(3m + 2) \cdot \tau_{\text{ElGamal}}$  bits of communication.

**Path evaluation using a garbled decision tree ( $\text{Path}_G$ , Fig. 11).** The method for garbled decision tree (GDT) evaluation is analogous to Yao’s garbled circuit technique [Yao86] described in §2.2, but  $\text{Eval}(GDT, \text{in})$  returns the output classification in the clear. Our description is similar to that of [BPSW07] that has been improved in [Sch08, BFK<sup>+</sup>09, BFL<sup>+</sup>11]. It relies on the idea of encrypting two keys at each decision node, one to the left and one to the right child nodes, along with their node indices, such that the evaluator  $\mathcal{C}$  cannot deviate from the path corresponding to the comparison results with his input vector  $\mathbf{x}$ . We introduce *dummy decision nodes* for hiding the length of the path to the classification node. Revealing this length would reveal information about the topology of the tree, especially when multiple protocol runs with the same model are possible. Alternatively, one can use full decision trees [BPSW07], but then the overhead is exponential in the depth  $d$ . This is much higher for large trees than using depth-padding with  $m'$  decision nodes (cf. §2.1) as shown in Fig. 12. Moreover, we re-

Server $\mathcal{S}$		Client $\mathcal{C}$	
Input: $\mathbf{a} = (a_1, \dots, a_m)$ , public key $pk$		Input: $\mathbf{b} = (b_1, \dots, b_m)$ , key pair $pk, sk$	
Output: $\perp$		Output: classification $v$	
For $j \in \{1, \dots, m+1\} : r_j \leftarrow_R \mathbb{Z}_p^*, r'_j \leftarrow_R \mathbb{Z}_p^*$			
For $i \in \{1, \dots, m\}$ $[B_i] \leftarrow [b_i \oplus a_i], [\mathbf{ec}_{i,0}] \leftarrow [B_i], [\mathbf{ec}_{i,1}] \leftarrow [1 - B_i]$		For $i \in \{1, \dots, m\} : [b_i] \leftarrow \text{Enc}(b_i)$	
For $j \in \{1, \dots, m+1\}$ $[\mathbf{pc}_j] \leftarrow [\sum_{e_{j,k} \in P} \mathbf{ec}_{j,k}], [\overline{\mathbf{pc}}_j] \leftarrow [r_j \cdot \mathbf{pc}_j], [\overline{v}_j] \leftarrow [r'_j \cdot \mathbf{pc}_j + v_j]$		Permuted $[\overline{\mathbf{pc}}_j], [\overline{v}_j]$	
		For $j \in \{1, \dots, m+1\}$ If $\text{Dec}_{sk}([\overline{\mathbf{pc}}_j]) = 0$ then <b>Return</b> $v \leftarrow \text{Dec}_{sk}([\overline{v}_j])$	

Fig. 10. Path<sub>H</sub> – Oblivious path evaluation using additively homomorphic encryption from [TMZC17].

Server $\mathcal{S}$		Client $\mathcal{C}$	
Input: $\mathbf{k}^0 = (k_1^0, \dots, k_{m'}^0), \mathbf{k}^1 = (k_1^1, \dots, k_{m'}^1)$		Input: $\mathbf{k} = (k_1, \dots, k_{m'})$	
Output: $\perp$		Output: classification $v$	
$\overline{DT} \leftarrow \text{DepthPad}(DT), GDT \leftarrow \text{Garble}(\overline{DT}, \mathbf{k}^0, \mathbf{k}^1)$		$GDT$	Offline
		<b>Return</b> $v \leftarrow \text{Eval}(GDT, \mathbf{k})$	Online

Fig. 11. Path<sub>G</sub> – Oblivious path evaluation using a garbled decision tree.

veal only the depth-padded number of decision nodes  $m'$ , and nothing about  $m$  since both of them would reveal information about the tree topology.

The server randomly permutes the nodes and garbles the decision tree so that each node has an index  $i$  and an encryption key  $k_i$  corresponding to it. At each node, the keys and indices of the child nodes are stored in an encrypted manner using the key of the node and keys  $k_i^0$  or  $k_i^1$  depending on the comparison result. This means that at node  $i$ ,  $\text{Enc}_{k_i, k_i^0}(k_l || l)$  and  $\text{Enc}_{k_i, k_i^1}(k_r || r)$  are stored, where  $l$  is the index of the node on the left and  $r$  is the index of the node on the right. The order of the two ciphertexts is permuted according to the color bits of  $k_i^0$  and  $k_i^1$ . The server sends  $GDT$  to the client, who can evaluate it by decrypting one path leading to the leaf node storing the classification. The communication in this protocol is sending the  $GDT$ , each node of which stores two  $\tau_{\text{sym}}$ -bit values and two indices, i.e.,  $2m' \cdot (\tau_{\text{sym}} + \log_2(m' + 1))$ . An extension of this protocol presented in §4.2 reduces the client's computation to  $\mathcal{O}(dt)$  symmetric key operations.

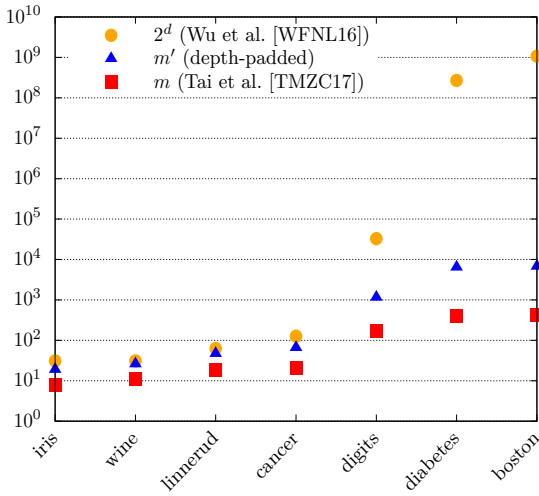
## 4 Protocols for Private Decision Tree Evaluation

In this section, we discuss all possible combinations of the sub-protocols from §3 that result in private decision tree evaluation protocols. We assess these in Tab. 4, which is an extended version of Tab. 1. In addition to the sub-protocols, we recapitulate the number of online rounds and the leakage about the model. In private decision tree evaluation, the client only learns the result and some information about the size of the server's decision tree: either the number of decision nodes  $m$  or the number of padded decision nodes  $m'$  and potentially the depth  $d$ . We adapt all resulting protocols with security against passive adversaries to an offline-online setting. We note that it is possible to precompute parts of the protocols even if not shown in the respective figures. In sub-protocols based on homomorphic encryption, we can precompute encryptions independent from the client's input, while in sub-protocols based on garbling techniques, we can perform the base OTs and precompute OTs (cf. §2.2).



Protocol	$\bar{m}$	Selection (Sel <sub>*</sub> )	Interface	Comparison (Comp <sub>*</sub> )	Interface	Path evaluation (Eval <sub>*</sub> )	Online rounds	Leakage
[BPSW07]	$m$	H		G		G	4	$m, d_{\text{path}}$
[WFNL16]	$m$	H (Fig. 8)		H (Fig. 8)		H	6	$m$
HGG [BFK <sup>+</sup> 09]	$m'$	H (Fig. 6)	G (Fig. 13)	G (Fig. 9)		G (Fig. 11)	4	$m', d$
GGG [BFK <sup>+</sup> 09]	$m'$	G (Fig. 7)		G (Fig. 9)		G (Fig. 11)	2	$m', d$
HHH [TMZC17]	$m$	H (Fig. 8)		H (Fig. 8)		H (Fig. 10)	4	$m$
HHG	$m'$	H (Fig. 8)		H (Fig. 8)	G (Fig. 14)	G (Fig. 11)	4	$m', d$
HGH	$m$	H (Fig. 6)	G (Fig. 13)	G (Fig. 9)		H (Fig. 10)	6	$m$
GGH	$m$	G (Fig. 7)		G (Fig. 9)		H (Fig. 10)	4	$m$

**Table 4.** Concrete protocols based on the presented sub-protocols for selection, comparison and path evaluation, where  $\bar{m}$  denotes either the number of decision nodes  $m$  or padded decision nodes  $m'$  in the protocols. Round complexity and leakage of different private decision tree evaluation protocols based on homomorphic encryption H and garbling techniques G.



**Fig. 12.** Number of decision nodes in the different protocols for the decision trees trained on real-world datasets from Tab. 3. Note that the  $y$ -axis is in logarithmic scale.

*Security guarantees.* We combine sub-protocols secure against passive adversaries with secret shared output between the two parties. The outputs of our primitives for selection (Sel<sub>H</sub> and Sel<sub>G</sub>) and comparison (Comp<sub>H</sub> and Comp<sub>G</sub>) as well as the inputs for comparison and path evaluation (Path<sub>H</sub> and Path<sub>G</sub>) are secret shared as indicated in Figs. 3-5, i.e., the parties do not learn any information by themselves. The secret sharing technique is either XOR-sharing, additive blinding or garbled inputs/outputs with the corresponding key(s). Therefore, our sequentially composed protocols remain secure against passive adversaries.

*Selection and Comparison Strategy.* For oblivious selection and comparison, the state-of-the-art protocols use additively homomorphic encryption Sel<sub>H</sub> (Fig. 6) [BFK<sup>+</sup>09] and Comp<sub>H</sub> (Fig. 8) [TMZC17]. This strategy is beneficial when it comes to communication and storage, but implies an increased runtime compared

to garbling techniques. Garbling techniques have been used for oblivious selection Sel<sub>G</sub> (Fig. 7) [BFK<sup>+</sup>09] and comparison Comp<sub>G</sub> (Fig. 9) [BFK<sup>+</sup>09]. Thereafter, these techniques have been claimed to be inefficient in [WFNL16, TMZC17]. However, our experiments in §5 show that they have a very efficient online phase, and perform better than the method based only on homomorphic encryption [TMZC17] with respect to total runtime in most cases as well.

*Path Evaluation Strategy.* As discussed in §3.3, the path evaluation sub-protocol of [TMZC17] using homomorphic encryption via Path<sub>H</sub> (Fig. 10) depends only on the number of decision nodes  $m$  and is therefore especially efficient with sparse and large decision trees. When this sub-protocol is used,  $\bar{m} = m$  in all preceding sub-protocols. This has been a tremendous improvement for real-world decision trees over the path evaluation method of Wu et al. [WFNL16], which depends exponentially on the depth  $d$  of the tree, due to expanding the tree to a full decision tree.

As opposed to this, in Path<sub>G</sub> (Fig. 11) the paths need to have the same length in order to hide the tree topology. Therefore, the tree is depth-padded to  $\bar{m} = m'$  decision nodes.  $m'$  depends on the topology of the decision tree, i.e., the level of all leaves (cf. §2.1).

We depict in Fig. 12 the different numbers of decision nodes that are used in state-of-the-art protocols and our depth-padded approach, i.e., the original number of decision nodes  $m$  (Path<sub>H</sub>, Fig. 10, Tai et al. [TMZC17]), the depth-padded number of decision nodes  $m'$  (Path<sub>G</sub>, Fig. 11), and the number of decision nodes in a full tree  $2^d$  (Wu et al. [WFNL16]).

## 4.1 Protocol Combinations

In this section, we describe all protocols resulting from valid combinations of sub-protocols presented in §3, and adapt them to the offline-online setting. Here, all operations that depend only on the input of the server  $S$ 's decision tree are performed in the offline phase, whereas all operations that depend on the client  $C$ 's input are performed in the online phase. Protocols that are efficient in the online phase are especially important in applications where the client uses a computationally weak device. Most of the computation can be done offline, when the device is idle, and the query itself triggers the online phase, which is performed more efficiently. We show that protocols based on garbling techniques are especially efficient in this setting, whereas the sub-protocols based on homomorphic encryption require the input directly at the start and therefore only few of the computationally intense tasks can be performed offline. ABC denotes the protocol that uses  $\text{Sel}_A$  for selection,  $\text{Comp}_B$  for comparison and  $\text{Path}_C$  for path evaluation.

**HHH:  $\text{Sel}_H + \text{Comp}_H + \text{Path}_H$  [TMZC17].** The state-of-the-art protocol using only homomorphic encryption for all three sub-protocols was presented in [TMZC17], the comparison protocol of which was first used for private decision tree evaluation in [WFNL16]. We recapitulate it here as a combination of the homomorphic sub-protocol  $\text{Sel}_H + \text{Comp}_H$  (Fig. 8) with  $\text{Path}_H$  (Fig. 10). The advantage of this protocol is that very low communication is necessary between the two parties and the selection happens along with the comparison. However, most computationally expensive cryptographic operations must be computed online, since the computation depends on the client's input.

**HGH:  $\text{Sel}_H + \text{Comp}_G + \text{Path}_H$ .** An oblivious comparison protocol that can be used to output shares of the comparison bits is Yao's GC protocol is shown in  $\text{Comp}_G$  (Fig. 9). Here,  $m$  garbled comparison circuits can be precomputed in an offline phase. Thereafter, the client evaluates these in the online phase (using SIMD in parallel as described in §2.2) to retrieve the comparison color bits that are shares of the output bits. However, this comparison protocol requires a selection sub-protocol to choose  $m$  inputs to the  $m$  comparisons from the client's  $n \leq m$  inputs. In this protocol, we use  $\text{Sel}_H$  with Paillier or DGK encryption (cf. Fig. 6) in order to achieve linear complexity. When combining these approaches, the result of  $\text{Sel}_H$  is not the same as the input of  $\text{Comp}_G$ , so the interface described below and depicted in Fig. 13 is inserted to obliviously unblind

the output. After the comparison, the path evaluation  $\text{Path}_H$  of [TMZC17] can be used as before (cf. Fig. 10).

*Combining homomorphic selection and garbled comparison ( $\text{Sel}_H \rightarrow \text{Comp}_G$ , Fig. 13).* The result of  $\text{Sel}_H$  in Fig. 6 is a set of blinded  $(t + s)$ -bit values for the client  $C$ , and the blinding  $s$ -bit values for the server  $S$ . In order to perform the comparison on  $t$ -bit values,  $S$  and  $C$  must unblind the result using a GC for subtracting each random value obliviously [BFK<sup>+</sup>09]. The subtractions are performed on  $t$ -bit values, since the most significant  $s$  bits of the blinded values can be dropped. The keys corresponding to the result are used in  $\text{Comp}_G$  (cf. Fig. 9). Overall, the  $\bar{m}$  garbled subtraction circuits require  $2\bar{m}t \cdot \tau_{\text{sym}}$  bits offline communication,  $\bar{m}t \cdot \tau_{\text{sym}}$  bits for precomputing random OTs, and  $\bar{m}t(2\tau_{\text{sym}} + 1)$  bits online communication for the OTs.

**GGH:  $\text{Sel}_G + \text{Comp}_G + \text{Path}_H$ .** Instead of instantiating the selection with homomorphic encryption, one can use  $\text{Sel}_G$ , a selection network (with  $\mathcal{O}(mt \log m)$  complexity) using garbled circuits (cf. Fig. 7). Here, most of the selection phase can be computed offline due to precomputed OTs and the independence of the GC from the client's inputs. Then,  $\text{Comp}_G$  (Fig. 9) can be directly performed, and the path evaluation sub-protocol  $\text{Path}_H$  of [TMZC17] (Fig. 10) is used as before. The advantage of this protocol is that though the offline communication is higher than in the previous protocols due to the selection network with superlinear complexity, it performs well in the online phase and due to the path evaluation phase, it depends only on the number of decision nodes  $m$ , since depth-padding is not required.

**HGG:  $\text{Sel}_H + \text{Comp}_G + \text{Path}_G$  [BFK<sup>+</sup>09].** Homomorphic selection  $\text{Sel}_H$  (Fig. 6) and garbled comparison  $\text{Comp}_G$  (Fig. 9) (with the interface from Fig. 13 between) can also be combined with the path evaluation technique  $\text{Path}_G$  (Fig. 11). This induces overhead in the number of decision nodes, since the depth-padded decision tree has a larger number of internal nodes  $m' \geq m$  (cf. Fig. 12). However, this path evaluation protocol can more efficiently be transformed to an offline-online setting, since the GCs for comparison and the garbled decision tree can be sent offline.

**GGG:  $\text{Sel}_G + \text{Comp}_G + \text{Path}_G$  [BFK<sup>+</sup>09].** This protocol consists exclusively of sub-protocols that use garbling techniques: selection  $\text{Sel}_G$  (Fig. 7) and comparison  $\text{Comp}_G$  (Fig. 9) with garbled circuits, and garbled path evaluation  $\text{Path}_G$  (Fig. 11). It has the highest communication, though most data can be sent in the offline phase.

Another advantage of this protocol is that it can be easily extended to provide security against a mali-

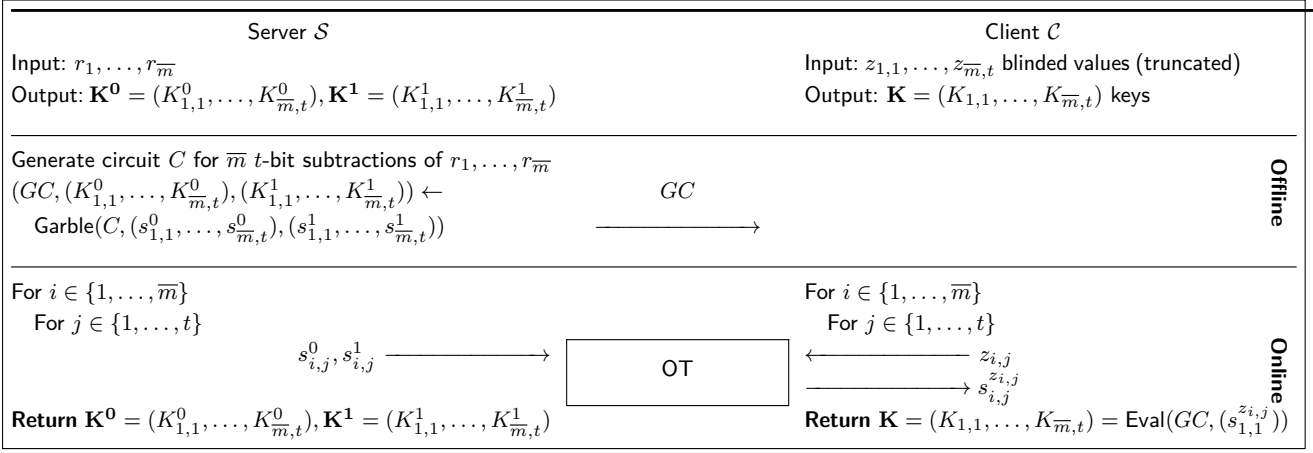


Fig. 13.  $\text{Sel}_H \rightarrow \text{Comp}_G$  – Interface between the instantiation for  $\text{Sel}_H$  (Fig. 6) and  $\text{Comp}_G$  (Fig. 9).

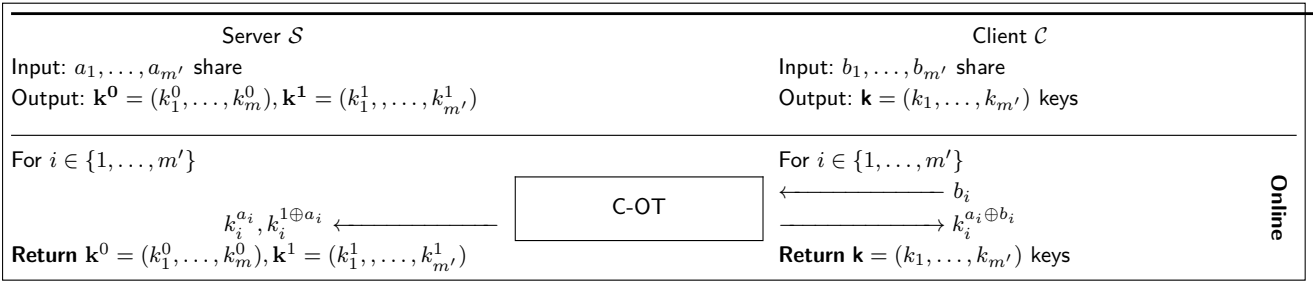


Fig. 14.  $\text{Comp}_H \rightarrow \text{Path}_G$  – Interface between the instantiation for  $\text{Comp}_H$  (Fig. 8) and  $\text{Path}_G$  (Fig. 11).

cious client: Since the only messages sent by the client are in the OTs, one can use OT extension with security against malicious clients, e.g., [ALSZ15, KOS15, ALSZ17], which are only slightly less efficient than OT extension with passive security. This adds little overhead, whereas securing the other solutions against malicious clients needs more expensive conditional OTs or zero-knowledge proofs, e.g., as for HHH [TMZC17].

**HHG:  $\text{Sel}_H + \text{Comp}_H + \text{Path}_G$ .** Due to the worse efficiency of the homomorphic selection and comparison  $\text{Sel}_H + \text{Comp}_H$  (Fig. 8) and the high number of decision nodes  $m'$  for the garbled path evaluation method  $\text{Path}_G$  (Fig. 11), this protocol combination is the least efficient. Moreover, it requires an interface of OTs described below and depicted in Fig. 14.

*Combining homomorphic comparison and garbled path evaluation ( $\text{Comp}_H \rightarrow \text{Path}_G$ , Fig. 14).* The result of  $\text{Sel}_H + \text{Comp}_H$  (Fig. 8) is for both parties a share of the comparison bit  $c_i = a_i \oplus b_i$  for all  $i \in \{1, \dots, m'\}$ , while the input to  $\text{Path}_G$  (Fig. 11) is a key for all comparison bits for the client  $\mathcal{C}$  and two keys for both possible bits for the server  $\mathcal{S}$ . After  $\mathcal{S}$  generates these keys,  $\mathcal{C}$  and  $\mathcal{S}$  engage in a 1-out-of-2 OT protocol for each comparison bit (cf. §2.2). This conversion requires  $m' \tau_{\text{sym}}$  bit offline and  $m'(\tau_{\text{sym}} + 1)$  bit online communication,

since we can use the correlated OT (C-OT) extension optimization of [ALSZ13].

## 4.2 Protocol Extensions

In this section, we describe protocol extensions, including a natural modification of the HGG and GGG protocols, which allows the client to perform less computation and evaluate the decision tree in a similar manner as in plain evaluation with only  $\mathcal{O}(dt)$  cryptographic operations. We compare this method with the protocol of [JS18] that has the same asymptotic complexity.

**Path evaluation with computationally restricted client.** A natural extension of HGG and GGG is to perform only the required  $d$  comparisons during the path evaluation phase as in the case of plain evaluation, instead of doing all  $m'$  comparisons in advance. This improves computation, though we cannot use SIMD operations for the comparisons anymore, since the GCs are evaluated sequentially along the evaluation path. Similarly, the client can decrypt only  $d$  (instead of  $m'$ ) homomorphically encrypted blinded values in HGG in  $\text{Sel}_H$  (cf. Fig. 6) before these comparisons. However, the communication remains unchanged, since the actual evaluation path taken needs to remain oblivious to the server.

Joye and Salehi present a similar protocol in [JS18]. In their protocol, full decision trees are utilized to hide the topology of the tree, and a 1-out-of- $2^i$  OT is performed at the  $i^{\text{th}}$  level ( $i \in \{1, \dots, d\}$ ). However, the full tree can be replaced with the depth-padded tree described in §2.1, in which case 1-out-of- $m'$  OTs are sufficient for levels where  $m' < 2^i$ . Though the communication of this protocol ( $\mathcal{O}(n+d(t+m'))$ ) is better than that of HGG ( $\mathcal{O}(n+m't)$ ) for small decision trees, its dependency on the depth makes it worse for larger examples such as the boston dataset (cf. §2.1). Moreover, it has  $\mathcal{O}(d)$  online rounds instead of the constant number of 4 rounds provided by HGG. Therefore, we conclude that our HGG protocol is more efficient in most scenarios.

**Other extensions.** In §B.1, we describe modifications for categorical variables, and in §B.2 for securely evaluating random forests. Classification confidence values (correctness probabilities), can be appended to the classification labels in a straightforward manner.

## 5 Performance Evaluation

In this section, we compare the runtime and communication of all resulting protocols from §4. In our implementation, we instantiate primitives corresponding to security level  $\kappa = 128$  (and  $\kappa = 112$  for public-key primitives) as recommended by NIST for use until 2030 [BBB<sup>+</sup>16], and statistical security parameter  $s = 40$ , and as in prior work, we set the bitlength to  $t = 64$ .  $\tau_*$  denotes the ciphertext sizes:  $\tau_{\text{sym}} = 128$  for symmetric-key encryption,  $\tau_{\text{Paillier}} = 4096$  for Paillier encryption,  $\tau_{\text{DGH}} = 2048$  for DGH encryption, and  $\tau_{\text{ElGamal}} = 514$  for ElGamal encryption with elliptic curves and point compression.

The underlying frameworks for our open-source implementation available at <https://encrypto.de/code/PDTE> are the ABY secure two-party computation framework [DSZ15]<sup>1</sup> and the mcl library<sup>2</sup> that includes a highly optimized lifted ElGamal implementation. ABY implements the state-of-the-art optimizations of Yao’s garbled circuit protocol described in §2.2. In addition, we implement an analogue technique to that of point-and-permute in order to avoid trial decryption of garbled nodes in oblivious path evaluation  $\text{Path}_G$  [Sch08]. We precompute OTs and homomorphic encryption when possible (i.e., Figs. 6, 7, 8, 10, 14). We give further de-

tails on our implementation in §C. We show the performance of the protocols for the datasets from Tab. 3 as well as for a full tree  $\text{full}(13)$  with  $d = n = 13$  for a comparison for dense trees.

**Communication.** The offline, online, and total communication of the protocols is given in Fig. 15a, Fig. 15b, and Fig. 15c, resp. We observe that garbling techniques allow for precomputation and offline communication, but require more communication in total. Homomorphic encryption-based methods have less communication, however, almost all expensive computation and all communication happen in the online phase. Methods using  $\text{Path}_H$  (i.e., GGH, HGH and HHH [TMZC17]) have a clear advantage for large sparse trees, which is lost in case extremely dense trees are considered (such as our example full tree  $\text{full}(13)$ ). For these kind of trees, Wu et al.’s protocol [WFNL16] has lower communication as discussed in §7.

**Runtime.** For our benchmarks we use two machines equipped with an Intel Core i7-4790 CPU @ 3.6 GHz and 32 GB of RAM that support Intel’s AES-NI for fast AES operations. Our benchmarks are run in a LAN setting with 1 Gbit/s and 0.5 ms latency. Runtimes are reported from an average of 10 executions. In our experiments, we neglect the costs for the base OTs and for generating the keypair for additively homomorphic encryption, since they are a one-time expense that can be re-used over multiple protocol executions. We also neglect the cost for trial decryption of the classification value in  $\text{Path}_H$ , and note that for a 16-bit value it is around 40 ms on average.

The offline runtimes are given in Fig. 16a, the online runtimes are given in Fig. 16b, while the total runtimes are depicted in Fig. 16c. We can see that homomorphic encryption-based methods ( $\text{Sel}_H$  and  $\text{Comp}_H$ ) perform an order of magnitude worse than their garbling technique-based alternatives ( $\text{Sel}_G$  and  $\text{Comp}_G$ ). However, we observe the advantage of the homomorphic encryption-based path evaluation technique ( $\text{Path}_H$ ), where the number of decision nodes  $m$  is unchanged, whereas in the path evaluation method using a garbled decision tree ( $\text{Path}_G$ ) depth-padding increases the number of decision nodes to  $m \leq m' \leq \frac{1}{2}m(m+1)$ , which can be significantly larger than  $m$  (cf. Fig. 12 and Tab. 3). Our protocol GGH has the fastest total and online runtime of below 1 second for our largest real-world example boston. For extremely dense decision trees such as our example full tree  $\text{full}(13)$ , this advantage is lost (since no padding is necessary) and the runtime of protocols based on garbling techniques is an order of mag-

<sup>1</sup> <https://github.com/encryptogroup/ABY>

<sup>2</sup> <https://github.com/herumi/mcl>

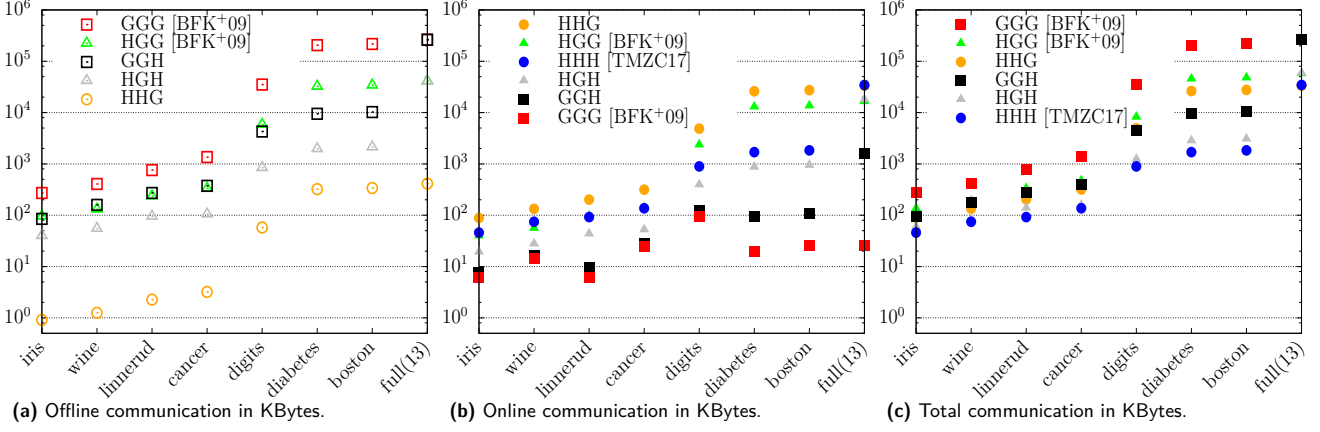


Fig. 15. Communication of protocols using the example datasets from Tab. 3. Note that the  $y$ -axis is in logarithmic scale.

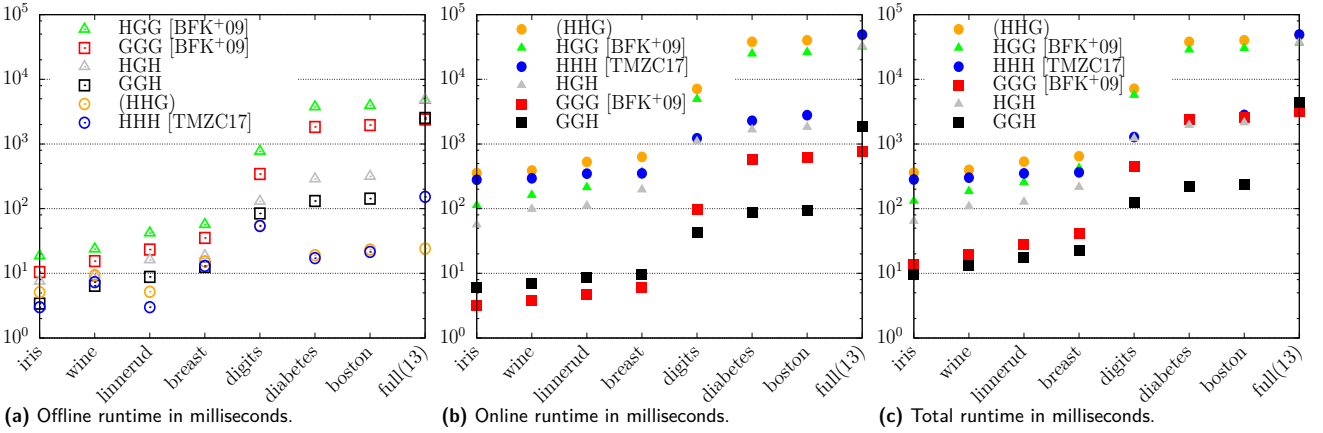


Fig. 16. Runtime of protocols using the example datasets from Tab. 3. Note that the  $y$ -axis is in logarithmic scale.

nitude lower than that of the protocols based on homomorphic encryption in the LAN setting.

**Tradeoff.** In order to show the communication ( $x$ -axis) vs. computation ( $y$ -axis) tradeoff of all resulting protocols, we give figures for decision trees trained on two example real-world datasets: a small dataset wine, and our largest dataset boston.

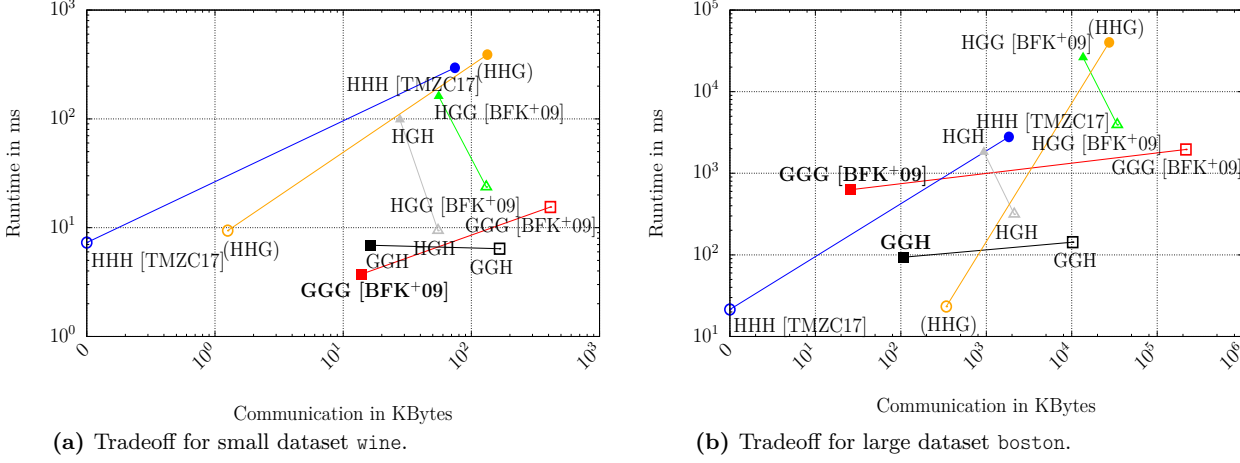
While the number of input features  $n$  is relatively small for all datasets, the number of (potentially padded) decision nodes significantly affects the efficiency of our sub-protocols. We depict in Fig. 17a and Fig. 17b the tradeoff between offline and online runtime and communication for all protocols for the wine and boston datasets, respectively, while Fig. 18a and Fig. 18b depict the corresponding total complexities.

The larger the dataset, the more dummy nodes are introduced during depth-padding (cf. §2.1) when Path<sub>G</sub> is used for path evaluation. Moreover, the selection network in Sel<sub>G</sub> has superlinear size  $\mathcal{O}(\overline{m} \log \overline{m})$ , which implies a significant overhead in offline communication. This gap can be observed on the figures: the

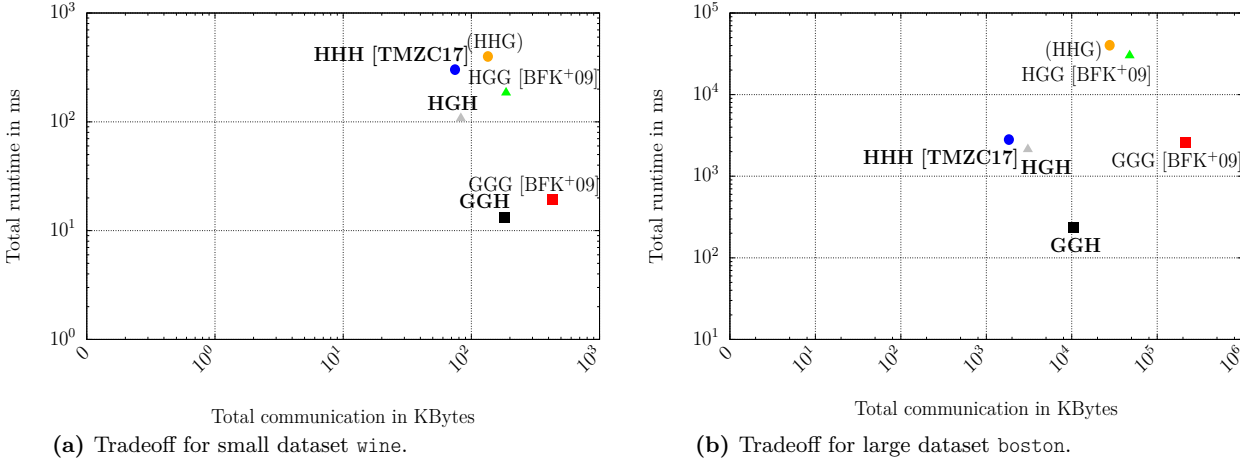
protocols using garbling techniques for path evaluation lose their performance advantage with growing number of dummy nodes, and the protocols using garbling techniques for selection become less practical due to the largely increased amount of offline communication for transmitting the selection network (which implies storage requirements as well). For instance, our largest decision tree trained on the boston dataset requires around 1.8 MBytes of total communication with HHH, 3 MBytes with HGH, 9 MBytes with GGH, and more than 200 MBytes with GGG. The designer of an application using private decision tree evaluation can consider these tradeoffs and choose the best suited protocol.

## 6 Concrete Improvements and Recommendations

We investigate the concrete tradeoffs between our identified hybrid protocols GGH and HGH from §4.1 and



**Fig. 17.** Tradeoff between communication ( $x$ -axis) and runtime ( $y$ -axis). The figures show the offline (unfilled squares) and online (filled squares) complexities (connected with a line). Note that both axes are in logarithmic scale, with an additional 0 on the  $x$  axis for depicting the offline communication of the HHH protocol. Pareto points, i.e., protocols where one property (computation or communication) cannot be improved without the other property becoming worse, for online complexities are marked in bold.



**Fig. 18.** Tradeoff between total communication ( $x$ -axis) and runtime ( $y$ -axis). Note that both axes are in logarithmic scale, with an additional 0 on the  $x$  axis for depicting the offline communication of the HHH protocol. Pareto points are marked in bold.

the state-of-the-art protocols HHH of [TMZC17] optimized for communication and GGG of [BFK<sup>+</sup>09] optimized for online computation. We show that the state-of-the-art protocols that exclusively use one of the two paradigms HHH with additively homomorphic encryption or GGG with garbling techniques can be replaced by our hybrid protocols HGH and GGH that provide a more reasonable tradeoff for larger decision trees.

In Figs. 17 and 18, we mark in bold pareto points for online and total tradeoffs, respectively, i.e., we bold the protocols where there is no possibility to improve one property without the other property becoming worse.

*Tradeoff with HHH [TMZC17].* The improvement in total and online runtime of GGH over HHH is more than an order of magnitude for our examples with increased total communication by 2-5x. HGH improves over HHH

in total and online runtime by 2-5x with only slightly increased total communication. The online communication is better for both GGH and HGH than for HHH.

*Tradeoff with GGG [BFK<sup>+</sup>09].* GGH improves over GGG for large datasets by an order of magnitude in all complexities due to the fact that it does not need any padding for the decision nodes. For small datasets, GGG is better than GGH in online runtime and communication by up to a factor of 2, but is worse in both total complexities. HGH improves even more (by 5-73x) for our datasets over the communication of GGG due to the difference in the number of decision nodes. The online runtimes of HGH are larger by 2-18x for real-world datasets, and the total runtimes are also larger by a factor of 5 for small datasets. However, it gets better with

larger (sparse) datasets due to the increasing number of decision nodes in the depth-padded tree.

**Recommendations.** We identify the following aspects to take into consideration when deciding which protocol to use for an application: the *dataset size*, and the boundaries on *communication and computational power* (e.g., network throughput and client storage capacity). When the client has high storage capacity, protocols GGG [BFK<sup>+</sup>09] or GGH from §4.1 provide the best online performance, depending on the dataset size. The former may have a slightly more efficient online phase, but higher communication due to its selection protocol (cf. §3.1). When the client is computationally bounded and has little storage capacity, our protocols HGH and GGH from §4.1 provide the best solutions.

The state-of-the-art protocol HHH of [TMZC17] has the lowest total communication, but uses computationally heavy public-key operations that depend almost entirely on the client’s input and hence do not allow for much offline precomputation. Our protocols HGH or GGH from §4.1 that combine its path evaluation with garbling techniques yield an order of magnitude faster runtimes while slightly increasing communication.

## 7 Related Work

Private decision tree evaluation was firstly considered in [BPSW07], with application to private evaluation of remote diagnostic programs. A protocol based on homomorphic encryption where the server evaluates a branching program on the client’s encrypted input was proposed in [IP07]. This has been improved in [BFK<sup>+</sup>09, BFL<sup>+</sup>11] where a protocol based on mainly symmetric-key operations was proposed. Bost et al. use additively homomorphic encryption to evaluate the decision tree expressed as a polynomial [BPTG15]. Recently, Wu et al. [WFNL16], Tai et al. [TMZC17] and Joye and Salehi [JS18] improved the state-of-the-art of private decision tree evaluation protocols. These works rely on additively homomorphic encryption using the Diffie-Hellman assumption and present protocols that achieve security against semi-honest adversaries or malicious clients. Tai et al. [TMZC17] eliminate the exponential dependency on the depth of the tree that was present in [WFNL16] by representing decision trees as linear functions. This implies enormous improvement when large decision trees are considered. These, in practice, are usually not very dense, and therefore, we use this protocol to instantiate HHH. Wu et al.’s proto-

col [WFNL16] would perform better than HHH for circuits with  $m \sim 2^d$  [TMZC17]. However, already for a very dense tree with  $m = 2^{d-2}$ , HHH has about half the communication compared to [WFNL16], whereas the necessary computation is around the same.

The protocols of [WFNL16] and [TMZC17] use the DGK comparison protocol [DGK07]. Joye and Salehi present an optimization on the DGK protocol and a private decision tree evaluation protocol with  $\mathcal{O}(d)$  rounds in [JS18] (cf. §3.2). For private decision tree evaluation with constant rounds, their optimization on the DGK protocol is not applicable. The alternative solution to the DGK protocol of [LZS18] uses fully homomorphic encryption and performs well for small bitlengths  $t$  of the features and inputs.

A different solution for evaluating full private decision trees using the so-called commodity-based model was proposed in [CDH<sup>+</sup>17], where correlated randomness is distributed by a trusted authority to the computing parties or pre-computed during the offline phase, which are used in the online phase.

In concurrent and independent related work Tuono et al. [TKK19] represent the decision tree as an array, and implement oblivious array indexing. For this, they use either garbled circuits, oblivious transfer or oblivious RAM (ORAM), the latter of which results in a protocol with sub-linear complexity in the size of the decision tree. Their protocols require  $dt$  comparisons as that of [JS18] and our protocol extension for GGG and HGG §4.2, but require at least  $\mathcal{O}(d)$  rounds of communication [TKK19, Tab. 2].

Alternatively, private decision tree evaluation can be solved using generic private function evaluation (PFE) protocols such as [KM11, MS13, MSS14]. However, these solutions utilize Boolean circuits as the underlying representation of the functionality. Transforming the decision tree into a Boolean circuit would imply additional unnecessary overhead on the size of the function (i.e.,  $\mathcal{O}(tm \log m)$  gates for the selection of inputs as in  $\text{Sel}_G$  in §3.1). Secure evaluation of universal circuits (UCs) is an equivalent solution for PFE [Val76, KS08b, KS16, GKS17]. UC- and OT-based PFE protocols [KS08b, MS13, KS16, GKS17] need  $\mathcal{O}((tm \log m) \log(tm \log m))$ , and HE-based protocols [KM11, MS13, MSS14] need  $\mathcal{O}(tm \log m)$  computation and communication. These complexities are larger than that of protocols designed specifically for DTs.

There are alternative approaches for classification based on machine learning, such as deep neural networks, that can be evaluated in a private manner [SS08, BFL<sup>+</sup>11, LJLA17, MZI17, RWT<sup>+</sup>18, JVC18, BFR<sup>+</sup>18,



BDK<sup>+</sup>18]. Private evaluation of machine learning models incur a natural overhead, but they can perform classifications of real-world datasets in the order of seconds.

## Acknowledgements

We thank the anonymous reviewers for their valuable feedback on the paper. This work was supported by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP, by the DFG as part of project E4 within the CRC 1119 CROSSING, by the Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS), and by Business Finland (CloSer project, 3881/31/2016).

## References

- [ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Computer and Communications Security (CCS'13)*, pages 535–548. ACM, 2013.
- [ALSZ15] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions with security for malicious adversaries. In *Advances in Cryptology – EUROCRYPT'15*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015.
- [ALSZ17] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer extensions. *J. Cryptology*, 30(3):805–858, 2017.
- [AM18] Bushra A. AlAhmadi and Ivan Martinovic. Malclassifier: Malware family classification using network flow sequence behaviour. In *2018 APWG Symposium on Electronic Crime Research (eCrime'18)*, pages 1–13. IEEE, 2018.
- [Bar15] Jeff Barr. Amazon machine learning – make data-driven decisions at scale. [aws.amazon.com/blogs/aws/amazon-machine-learning-make-data-driven-decisions-at-scale](https://aws.amazon.com/blogs/aws/amazon-machine-learning-make-data-driven-decisions-at-scale), 2015. Accessed: 2018-08-19.
- [BBB<sup>+</sup>16] Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid. Sp 800-57. recommendation for key management, part 1: General (revised). Technical report, Gaithersburg, MD, United States, 2016.
- [BDK<sup>+</sup>18] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *ACM Computer and Communications Security (CCS'18)*, pages 847–861. ACM, 2018.
- [Bea95] Donald Beaver. Precomputing oblivious transfer. In *Advances in Cryptology – CRYPTO'95*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.
- [BFK<sup>+</sup>09] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *European Symposium on Research in Computer Security (ESORICS'09)*, volume 5789 of *LNCS*, pages 424–439. Springer, 2009.
- [BFL<sup>+</sup>11] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-preserving ECG classification with branching programs and neural networks. *IEEE Transactions on Information Forensics and Security*, 6(2):452–468, 2011.
- [BFR<sup>+</sup>18] Ferdinand Brasser, Tommaso Frassetto, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, and Christian Weinert. Voiceguard: Secure and private speech processing. In *Annual Conference of the International Speech Communication Association (INTERSPEECH'18)*, pages 1303–1307. ISCA, 2018.
- [BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-

- key blockcipher. In *IEEE Symposium on Security and Privacy (S&P'13)*, pages 478–492. IEEE, 2013.
- [Big18] Inc. BigML. Machine learning made easy, beautiful and understandable. <https://bigml.com/>, 2018. Accessed: 2018-08-24.
- [BJL12] Dan Bogdanov, Roman Jagomägis, and Sven Laur. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Pacific Asia Workshop on Intelligence and Security Informatics (PAISI'12)*, volume 7299 of *LNCS*, pages 112–126. Springer, 2012.
- [BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *ACM Symposium on Theory of Computing (STOC'90)*, pages 503–513. ACM, 1990.
- [BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *ACM Computer and Communications Security (CCS'07)*, pages 498–507. ACM, 2007.
- [BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.
- [BS09] Justin Brickell and Vitaly Shmatikov. Privacy-preserving classifier learning. In *Financial Cryptography and Data Security (FC'09)*, volume 5628 of *LNCS*, pages 128–147. Springer, 2009.
- [BSR18] Diogo Barradas, Nuno Santos, and Luís Rodrigues. Effective detection of multimedia protocol tunneling using machine learning. In *USENIX Security Symposium'18*, pages 169–185. USENIX, 2018.
- [CDH<sup>+</sup>17] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson C. A. Nascimento, Wing-Sea Poon, and Stacey C. Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, To appear., 2017.
- [CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *Security and Cryptography for Networks (SCN'18)*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.
- [DCBA14] Manuel Fernández Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.
- [DGK07] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Australasian Conference on Information Security and Privacy (ACISP'07)*, volume 4586 of *LNCS*, pages 416–430. Springer, 2007.
- [DGK08] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*, 1(1):22–31, 2008.
- [DGK09] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. A correction to 'Efficient and secure comparison for on-line auctions'. *International Journal of Advanced Computer Technology (IJACT)*, 1(4):323–324, 2009.
- [DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography (PKC'01)*, volume 1992 of *LNCS*, pages 119–136. Springer, 2001.
- [DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.
- [ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology – CRYPTO'85*, volume 196 of *LNCS*, pages 10–18. Springer, 1985.
- [FW12] Pui Kuen Fong and Jens H. Weber-Jahnke. Privacy preserving decision tree learning using unrealized data sets. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):353–364, 2012.
- [GBC<sup>+</sup>97] Michael D. Garris, James L. Blue, Gerald T. Candela, Patrick J. Grother, Stanley Janet, and Charles L. Wilson. NIST form-based handprint recognition system (release 2.0). *Interagency/Internal Report (NISTIR) - 5959*, 1997.
- [GKG<sup>+</sup>18] Srishti Gupta, Abhinav Khattar, Arpit Gogia, Pon-nuram Kumaraguru, and Tanmoy Chakraborty. Collective classification of spam campaigners on Twitter: A hierarchical meta-path based approach. In *World Wide Web Conference on World Wide Web (WWW'18)*, pages 529–538. ACM, 2018.
- [GKS17] Daniel Günther, Ágnes Kiss, and Thomas Schneider. More efficient universal circuit constructions. In *Advances in Cryptology – ASIACRYPT'17*, volume 10625 of *LNCS*, pages 443–470. Springer, 2017.
- [HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security Symposium (NDSS'12)*. The Internet Society, 2012.
- [IK17] Aleksandar Ilic and Oleksandr Kuvshynov. Evaluating boosted decision trees for billions of users. <https://code.facebook.com/posts/975025089299409/evaluating-boosted-decision-trees-for-billions-of-users>, 2017. Accessed: 2018-08-19.
- [IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology – CRYPTO'03*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.
- [IP07] Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In *Theory of Cryptography Conference (TCC'07)*, volume 4392 of *LNCS*, pages 575–594. Springer, 2007.
- [IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *ACM Symposium on Theory of Computing (STOC'89)*, pages 44–61. ACM, 1989.
- [JS18] Marc Joye and Fariborz Salehi. Private yet efficient decision tree evaluation. In *Data and Applications Security and Privacy (DBSec'18)*, volume 10980 of

- LNCS, pages 243–259. Springer, 2018.
- [JSD<sup>+</sup>18] Mika Juuti, Sebastian Szyller, Alexey Dmitrenko, Samuel Marchal, and N. Asokan. PRADA: protecting against DNN model stealing attacks. *CoRR*, abs/1805.02628, 2018.
- [JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *USENIX Security Symposium’18*, pages 1651–1669. USENIX, 2018.
- [KM11] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In *Advances in Cryptology – ASIACRYPT’11*, volume 7073 of LNCS, pages 556–571. Springer, 2011.
- [KMAM18] Manish Kesarwani, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. Model extraction warning in mlaas paradigm. In *Annual Computer Security Applications Conference (ACSAC’18)*, pages 371–380. ACM, 2018.
- [KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology – CRYPTO’15*, volume 9215 of LNCS, pages 724–741. Springer, 2015.
- [KS08a] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages and Programming (ICALP’08)*, volume 5126 of LNCS, pages 486–498. Springer, 2008.
- [KS08b] Vladimir Kolesnikov and Thomas Schneider. A practical universal circuit construction and secure evaluation of private functions. In *Financial Cryptography and Data Security (FC’08)*, volume 5143 of LNCS, pages 83–97. Springer, 2008.
- [KS16] Ágnes Kiss and Thomas Schneider. Valiant’s universal circuit is practical. In *Advances in Cryptology – EUROCRYPT’16*, volume 9665 of LNCS, pages 699–728. Springer, 2016.
- [KSS09] Vladimir Kolesnikov, Ahmad-Reza Sadeghi, and Thomas Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *Cryptology and Network Security (CANS’09)*, volume 5888 of LNCS, pages 1–20. Springer, 2009.
- [Lic18] Moshe Lichman. UCI machine learning repository. <https://archive.ics.uci.edu/ml>. Irvine, CA: University of California, School of Information and Computer Science, 2018. Accessed: 2018-08-24.
- [LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM Computer and Communications Security (CCS’17)*, pages 619–631. ACM, 2017.
- [LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO’00*, volume 1880 of LNCS, pages 36–54. Springer, 2000.
- [LP02] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, 2002.
- [LZS18] Wenjie Lu, Jun-Jie Zhou, and Jun Sakuma. Non-interactive and output expressive private comparison from homomorphic encryption. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS’18)*, pages 67–74. ACM, 2018.
- [MAAG15] Michael J. Mayhew, Michael Atighetchi, Aaron Adler, and Rachel Greenstadt. Use of machine learning in big data analytics for insider threat detection. In *IEEE Military Communications Conference (MILCOM’15)*, pages 915–922. IEEE, 2015.
- [Mic18] Microsoft. Azure machine learning studio. <https://azure.microsoft.com/>, 2018. Accessed: 2018-08-24.
- [MLJ17] Inc. MLJAR. MLJAR: Machine learning for all. <https://mljar.com/>, 2016–2017. Accessed: 2018-08-24.
- [MS13] Payman Mohassel and Seyed Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *Advances in Cryptology – EUROCRYPT’13*, volume 7881 of LNCS, pages 557–574. Springer, 2013.
- [MSS14] Payman Mohassel, Seyed Saeed Sadeghian, and Nigel P. Smart. Actively secure private function evaluation. In *Advances in Cryptology – ASIACRYPT’14*, volume 8874 of LNCS, pages 486–505. Springer, 2014.
- [MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy (S&P’17)*, pages 19–38. IEEE, 2017.
- [NIZ<sup>+</sup>16] Ana Nika, Asad Ismail, Ben Y. Zhao, Sabrina Gaito, Gian Paolo Rossi, and Haitao Zheng. Understanding and predicting data hotspots in cellular networks. *Mobile Networks and Applications (MONET)*, 21(3):402–413, 2016.
- [OTGM18] Rebekah Overdorf, Carmela Troncoso, Rachel Greenstadt, and Damon McCoy. Under the underground: Predicting private interactions in underground forums. *CoRR*, abs/1805.04494, 2018.
- [Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology – EUROCRYPT’99*, volume 1592 of LNCS, pages 223–238. Springer, 1999.
- [PMG<sup>+</sup>17] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS’17)*, 2017, pages 506–519. ACM, 2017.
- [RG16] Carl Rabeler and Craig Guyer. Microsoft decision trees algorithm. <https://docs.microsoft.com/en-us/sql/analysis-services/data-mining/microsoft-decision-trees-algorithm>, 2016. Accessed: 2018-08-19.
- [RMD18] Alejandro Rago, Claudia Marcos, and J. Andres Diaz-Pace. Using semantic roles to improve text classification in the requirements domain. *Language Resources and Evaluation*, 52(3):801–837, 2018.
- [RWT<sup>+</sup>18] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS’18)*, pages

- 707–721. ACM, 2018.
- [Sch08] Thomas Schneider. Practical secure function evaluation. Master's thesis, Friedrich-Alexander University Erlangen-Nürnberg, Germany, February 27, 2008.
- [Ser18] Amazon Web Services. Data privacy. <https://aws.amazon.com/compliance/data-privacy-faq>, 2018. Accessed: 2018-08-19.
- [sld17] scikit-learn developers. scikit-learn – machine learning in python. <http://scikit-learn.org/stable/modules/tree.html>, 2017. Accessed: 2018-08-22.
- [SS08] Ahmad-Reza Sadeghi and Thomas Schneider. Generalized universal circuits for secure evaluation of private functions with application to data classification. In *Information Security and Cryptology (ICISC'08)*, volume 5461 of LNCS, pages 336–353. Springer, 2008.
- [SSSS17] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (S&P'17)*, pages 3–18. IEEE, 2017.
- [TASF09] Ajay Kumar Tanwani, M. Jamal Afridi, M. Zubair Shafiq, and Muddassar Farooq. Guidelines to select machine learning scheme for classification of biomedical datasets. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO'09)*, volume 5483 of LNCS, pages 128–139. Springer, 2009.
- [Ten98] Michel Tenenhaus. *La régression PLS: théorie et pratique*. Editions technip, 1998.
- [TKK19] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. Private evaluation of decision trees using sublinear cost. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2019(1):266–286, 2019.
- [TMZC17] Raymond K. H. Tai, Jack P. K. Ma, Yongjun Zhao, and Sherman S. M. Chow. Privacy-preserving decision trees evaluation via linear functions. In *European Symposium on Research in Computer Security (ESORICS'17)*, volume 10493 of LNCS, pages 494–512. Springer, 2017.
- [TZJ<sup>+</sup>16] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security Symposium'16*, pages 601–618. USENIX, 2016.
- [Val76] Leslie G. Valiant. Universal circuits (preliminary report). In *ACM Symposium on Theory of Computing (STOC'76)*, pages 196–203. ACM, 1976.
- [VC05] Jaideep Vaidya and Chris Clifton. Privacy-preserving decision trees over vertically partitioned data. In *Data and Applications Security (DBSec'05)*, volume 3654 of LNCS, pages 139–152. Springer, 2005.
- [VCKP08] Jaideep Vaidya, Chris Clifton, Murat Kantarcioglu, and A. Scott Patterson. Privacy-preserving decision trees over vertically partitioned data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2(3):14:1–14:27, 2008.
- [Wak68] Abraham Waksman. A permutation network. *Journal of the ACM*, 15(1):159–163, 1968.
- [WFNL16] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. Privately evaluating decision trees and random forests. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2016(4):335–355, 2016.
- [WGC18] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: Efficient and private neural network training. *Cryptology ePrint Archive*, 2018/442, 2018.
- [Wis18] Wise.io. Machine learning for the industrial internet of things. wise.io, 2018. Accessed: 2018-08-24.
- [Yao82] Andrew C.-C. Yao. Protocols for secure computations (extended abstract). In *Foundations of Computer Science (FOCS'82)*, pages 160–164. IEEE, 1982.
- [Yao86] Andrew C.-C. Yao. How to generate and exchange secrets (extended abstract). In *Foundations of Computer Science (FOCS'86)*, pages 162–167. IEEE, 1986.
- [YGL17] Fengpeng Yuan, Xianyi Gao, and Janne Lindqvist. How busy are you?: Predicting the interruptibility intensity of mobile users. In *Conference on Human Factors in Computing Systems (CHI'17)*, pages 5346–5360. ACM, 2017.
- [ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT'15*, volume 9057 of LNCS, pages 220–250. Springer, 2015.

## A Complexities of Sub-protocols

In this section, we describe the complexities of all sub-protocols described in §3. Note that oblivious transfers and homomorphic encryptions are precomputed in the offline phase when possible.

**Communication.** The communication of each sub-protocol is presented in Tab. 5. For all phases we observe a direct tradeoff: The homomorphic encryption-based methods have lower communication, while in case of garbling techniques, most communication can be shifted offline, so these achieve the lowest online communication.

**Computation.** The asymptotic computational complexity of each sub-protocol is shown in Tab. 6. We observe that most computationally intense tasks can be shifted offline for the protocols based on garbling techniques, while almost all operations are performed online when using homomorphic encryption. Unfortunately, even when using garbling techniques, the client needs to perform most cryptographic operations online, since he plays the role of the evaluator (cf. §2.2). In these techniques, however, the client requires less computational power than using homomorphic encryption.

Sub-protocol	Fig.	Offline	Online
Sel <sub>H</sub> Paillier	Fig. 6	-	$(n + \frac{\bar{m}}{t+s}) \cdot \tau_{\text{Paillier}}$
Sel <sub>H</sub> DGK	Fig. 6	-	$(n + \bar{m}) \cdot \tau_{\text{DGK}}$
Sel <sub>G</sub>	Fig. 7	$(n + 2S_{\bar{m} \geq n}^n)t \cdot \tau_{\text{sym}}$	$nt(2\tau_{\text{sym}} + 1)$
Sel <sub>H</sub> + Comp <sub>H</sub>	Fig. 8	-	$(n + \bar{m})t \cdot \tau_{\text{ElGamal}}$
Comp <sub>G</sub>	Fig. 9	$2\bar{m}t \cdot \tau_{\text{sym}}$	-
Path <sub>H</sub>	Fig. 10	-	$(3m + 2) \cdot \tau_{\text{ElGamal}}$
Path <sub>G</sub>	Fig. 11	$2m' \cdot (\tau_{\text{sym}} + \log_2(m' + 1))$	-
Sel <sub>H</sub> → Comp <sub>G</sub>	Fig. 13	$3\bar{m}t \cdot \tau_{\text{sym}}$	$\bar{m}t(2\tau_{\text{sym}} + 1)$
Comp <sub>H</sub> → Path <sub>G</sub>	Fig. 14	$m' \cdot \tau_{\text{sym}}$	$m'(\tau_{\text{sym}} + 1)$

**Table 5.** Offline and online communication of all sub-protocols, where  $n$ ,  $t$ ,  $m$ ,  $m'$  and  $\bar{m}$  denote the number input features, the bitlength of the features and thresholds, the number of decision nodes, padded decision nodes and either of the former two, respectively.  $S_{\bar{m} \geq n}^n$  denotes the size of the selection network that selects  $\bar{m}$  bits from  $n$  bits.  $\tau_{\text{sym}}$ ,  $\tau_{\text{DGK}}$ ,  $\tau_{\text{Paillier}}$  and  $\tau_{\text{ElGamal}}$  denote the size of the ciphertexts in the respective encryption schemes. Note that  $m \leq m' \leq \frac{1}{2}m(m+1)$ .

Sub-protocol	Fig.	Offline Server	Offline Client	Online Server	Online Client
Sel <sub>H</sub> Paillier	Fig. 6	$\mathcal{O}(\bar{m})\nu_{\text{Paillier}}$	-	$\mathcal{O}(\bar{m})\nu_{\text{Paillier}}$	$\mathcal{O}(n + \bar{m})\nu_{\text{Paillier}}$
Sel <sub>H</sub> DGK	Fig. 6	$\mathcal{O}(\bar{m})\nu_{\text{DGK}}$	-	$\mathcal{O}(\bar{m})\nu_{\text{DGK}}$	$\mathcal{O}(n + \bar{m})\nu_{\text{DGK}}$
Sel <sub>G</sub>	Fig. 7	$\mathcal{O}(t(\bar{m} \log \bar{m} + n))\nu_{\text{sym}}$	$\mathcal{O}(nt)\nu_{\text{sym}}$	-	$\mathcal{O}(t\bar{m} \log \bar{m})\nu_{\text{sym}}$
Sel <sub>H</sub> + Comp <sub>H</sub>	Fig. 8	-	$\mathcal{O}(nt)\nu_{\text{ElGamal(off)}}$	$\mathcal{O}(t\bar{m})\nu_{\text{ElGamal}}$	$\mathcal{O}(\bar{m}t)\nu_{\text{ElGamal}} + \mathcal{O}(nt)\nu_{\text{ElGamal(on)}}$
Comp <sub>G</sub>	Fig. 9	$\mathcal{O}(\bar{m}t)\nu_{\text{sym}}$	-	-	$\mathcal{O}(\bar{m}t)\nu_{\text{sym}}$
Path <sub>H</sub>	Fig. 10	-	$\mathcal{O}(m)\nu_{\text{ElGamal(off)}}$	$\mathcal{O}(md)\nu_{\text{ElGamal}}$	$\mathcal{O}(m)\nu_{\text{ElGamal}} + \mathcal{O}(m)\nu_{\text{ElGamal(on)}}$
Path <sub>G</sub>	Fig. 11	$\mathcal{O}(m')\nu_{\text{sym}}$	-	-	$\mathcal{O}(m')\nu_{\text{sym}}$
Sel <sub>H</sub> → Comp <sub>G</sub>	Fig. 13	$\mathcal{O}(\bar{m}t)\nu_{\text{sym}}$	$\mathcal{O}(\bar{m}t)\nu_{\text{sym}}$	-	$\mathcal{O}(\bar{m}t)\nu_{\text{sym}}$
Comp <sub>H</sub> → Path <sub>G</sub>	Fig. 14	$\mathcal{O}(m')\nu_{\text{sym}}$	$\mathcal{O}(m')\nu_{\text{sym}}$	-	-

**Table 6.** The number of cryptographic operations in the offline and online phases of all sub-protocols, with notations as in Tab. 5.

$\nu_{\text{sym}}$ ,  $\nu_{\text{DGK}}$ ,  $\nu_{\text{Paillier}}$  and  $\nu_{\text{ElGamal}}$  denote the cost of an operation (i.e., encryption, decryption or addition) in symmetric, DGK, Paillier and lifted ElGamal encryption, respectively.  $\nu_{\text{ElGamal(off/on)}}$  denote the offline/online costs of precomputed ElGamal encryption.

## B Extensions

In this section, we present extensions to Comp<sub>G</sub> and Path<sub>G</sub> presented in §3, which directly apply to the resulting protocols that use those as sub-protocols in §4.

### B.1 Categorical Variables

Since not every member of a feature vector is a numerical variable, we discuss the useful extension of the comparison phase in order to add support for categorical variables. In fact, a considerable portion of real datasets are a mixture of numerical and categorical variables. We measured the ratio of datasets containing categorical variables in the UCI machine learning repository [Lic18], and found that 33% of datasets contain categorical variables. For instance, a feature vector for the breast cancer dataset has the feature breast quadrant that specifies the position of the tumor in the patient's breast, i.e. it takes one of the following five values: left-up, right-up, left-low, right-low or

central. At the end of the protocol, the patient's profile is classified as no-recurrence-event or recurrence-event that determines the possibility of cancer recurrence.

Given categorical variable  $x_j \in U = \{u_1, \dots, u_{|U|}\}$ , the decision criteria is a set membership test of the form  $x_j \in U'$ , for some  $U' \subset U$  ( $j \in \{1, \dots, n\}$ ). Wu et al. supply the Sel<sub>H</sub> + Comp<sub>H</sub> subprotocol with the extension to handle categorical variables in [WFNL16]. We briefly describe a private approach to evaluate the decision nodes that operate over categorical variables when using Comp<sub>G</sub>. We notice that for testing the set inclusion, the server can create the  $|U|$ -bit masking value **Mask** based on  $U'$  such that  $\text{Mask}_i = 1$  if  $u_i \in U'$ , and 0 otherwise. This representation was used for private set intersection (PSI) in [HEK12]. Accordingly, for  $x_j = u_i$ , the client inputs the transformed  $|U|$ -bit variable  $x'_j$  which has only one *set* bit at position  $i$ . Then the original decision criteria  $x_j \in U'$  can be evaluated by the Boolean circuit which implements  $x'_j \wedge \text{Mask} = x'_j$ .

## B.2 Random Forests

Random forests improve the accuracy of decision trees by aggregating the result of  $n \geq 2$  decision trees trained on different random subsets of features. Before evaluating the  $n$  decision trees, in order to hide all information about the subset of features, all decision tree evaluations need to look the same from the client's point of view. Therefore, dummy comparisons are necessary to pad the decision trees to the maximal  $m$  or  $m'$  value.

Additionally, the aggregation step depending on the aggregation function has to be implemented. Wu et al. [WFNL16] provide a method that allows for any affine function of the results using additive secret sharing, which can be generalized to  $\text{Path}_H$  (cf. Fig. 10) of [TMZC17]. For  $\text{Path}_G$  (cf. Fig. 11) [BFK<sup>+</sup>09], instead of returning the results to the client, he can obtain keys corresponding to these. These can then be used to securely evaluate the GC corresponding to the aggregation function if the server uses these keys to generate it. The complexity depends on the aggregation function.

**Path<sub>H</sub> (Fig. 10).** We implement the path evaluation phase using lifted EC-ElGamal encryption over the same curve and library as for  $\text{Sel}_H + \text{Comp}_H$ .

**Path<sub>G</sub> (Fig. 11).** We implement garbled decision tree path evaluation in the ABY framework using two AES encryptions with AES-NI per decision node.

## C Implementation Choices

We describe our implementation choices used in §5.

**Sel<sub>H</sub> (Fig. 6).** The ABY framework includes an implementation for both Paillier and DGK encryptions (cf. §2.2). The ciphertext lengths are  $\tau_{\text{Paillier}} = 4096$  and  $\tau_{\text{DGK}} = 2048$ . However, Paillier encryption allows for ciphertext packing which reduces the number of ciphertexts and decryptions. We utilize ABY to build two versions of  $\text{Sel}_H$ , and conclude the advantage of the Paillier encryption both in runtime and communication.

**Sel<sub>G</sub> (Fig. 7).** We implement the garbled selection network introduced in [KS08a] in ABY.

**Sel<sub>H</sub> + Comp<sub>H</sub> (Fig. 8).** We use the `mcl` library to implement protocols using lifted ElGamal encryption over elliptic curve (EC) `secp256k1` with 256-bit key as Tai et al. [TMZC17]. This library has a highly optimized lifted EC-ElGamal implementation and supports point compression, i.e., an elliptic curve point can be expressed in  $256 + 1$  bits. Since an ElGamal ciphertext consists of two EC points,  $\tau_{\text{ElGamal}} = 514$ . Our baseline implementation is that of [LZS18].

**Comp<sub>G</sub> (Fig. 9).** We use comparison circuits in ABY using SIMD operations to enhance the performance of this sub-protocol.